Software *Patterns and Architecture*  
Under Examination Hammer  
An Approach to the Consolidation of Interdisciplinary Knowledge

A Thesis Submitted for the Degree of  
Doctor of Philosophy  
of the  
Australian National University

Hassan Almari  
29 November 2018
I declare that the work in this thesis is entirely my own and that to the best of my knowledge it does not contain any materials previously published or written by another person except where otherwise indicated.

Hassan Almari
29 November 2018
Special Thanks

To all my family members who gave me moral support and encouragement in the entire process of this study; utmost appreciation is extended to my parents, who gave me courage and laid a firm foundation for my education and success. I also appreciate my brothers and sisters for their ultimate encouragement and care. Thanks go, also, to my sons for their perseverance whilst I was away or busy. They missed out on paternal love. Last, I admire Jamilh's (my lovely wife's) positive attitudes and for being at the bedrock of every success I have ever achieved. When I was down and out, or in despair, she motivated me, whilst ensuring that I achieved what I had always expected to accomplish.
Acknowledgements

I take this opportunity to highly praise my remarkable supervisor, Dr. Clive Boughton, for his continuous support, advice, friendship, and patience. He has acted as my mentor, guiding me through the rugged terrain of academia, starting with my master’s program in 2008. He has provided me with his support, not only in regard to my studies, but also in regard to my work and personal matters. Essentially, it was his unrelenting efforts in providing feedback whenever needed, which enabled me to come up with a well-crafted project.

I would also like to recognise the efforts made by Dr. Henry Gardener and Dr. Athman Bouguettaya for their support and help during my surveys/distribution processes within this project.

Throughout this journey, there is also a number of people who provided me with guidance and ideas. To name but a few, I wish to salute my friends Dr. Carol Boughton for her editing and advice, Dr. Shayne Flint for being a wise and helpful friend during my work, and Dr. Ziyad Alshaikh for his recommendations during my first year of this research.

Equally, I salute all the people within the Saudi Ministry of Defence for their support during my research, especially Saudi Air Force for their hospitality and willingness to share their knowledge with me regarding software development.

My research has been a long journey, which has been full of obstacles; therefore, it would be disappointing if I fail to mention Dr. Sergey Denisov for his advice and professional help during this thesis’ formatting; Dr. Vashkar Chowdhury for his statistical advice and statistics proofreading; Ms Shareen and Mr. Angelo Victor Mercure for their overall proofreading; and Dr. Jasur Abdullaev for his advice, and support during the final phase of this thesis.

I wish to state that my transitional endeavour, from the military environment to education, has not been a bed of roses, as it has been full of ups and downs. Nevertheless, the experience and knowledge I have gained motivated me to appreciate Australian National University for giving me the opportunity to conduct my research. Specifically, I appreciate my fellow graduate students Mazen Alismail, and Dr Alexander Krumphols for their excellent discussions, which opened new ideas and possibilities for me. I cannot finish without mentioning members of staff working in the Research School of Computer Science for their support and guidance.

I cannot forget to mention my sponsor, the Royal Saudi Air Defence Force (RSADF) for their scholarship and support. I also salute the Saudi Arabian Embassy and Cultural Mission for their cooperation while conducting my thesis. Equally, I recognise, Ambassador Hassan Nazer (retired) and the new Saudi Ambassador Mr. Nabil AL SALEH for their personal care, and unlimited support during this journey.
Abstract

Software engineering is normally perceived, and even defined, based upon applicability of scientific and technical knowledge, in order to provide solutions to different challenges. The bright side of engineering concepts in general, is the continuous process of acquiring knowledge and skills needed to develop and make adjustments to various systems, in respect to helping humankind.

An important phase of this process is "Architecting", which is the big picture of any intended systems. While good architecture leads to successful systems, bad architecture can result in misfortune.

In this thesis, my proposition is to investigate, in depth, both theoretical (academic) and industry domains, regarding the way in which they treat Software Pattern (SP), Software Architecture (SA), and Software Architecture Evaluation (SAE) techniques.

I argue that the process of creating, evaluating, and documenting SPs and SA with no common guidelines, standards, and frameworks, will result in unused and conflicted information within their areas, which finally will impact the software engineering field. While the employment of interdisciplinary knowledge (such as SPs, modelling techniques, description languages, evaluation methods, standards, and frameworks), could elevate SA development and validation methodologies, and increase its utilisation within the software engineering community.

The goal here is to help build better systems, which could be improved by developing suitable SA, and evaluate its qualities by proper methods and tools, before further development, which should save time as well as money.

Therefore, after a long process of analysing the current-state-of-the-art, I have introduced in this thesis novel findings concerning descriptions, relationships, documentation, and utilisation in relation to SA, SAE, and SPs, through employing several investigatory techniques, including comparisons between reliable references, questionnaires, field study, and case study.

The investigation of SPs resulted in creating a database as a partial solution, in order to minimise their confusion within the literature, concerning their definitions, categorisations, and relationships with different quality attributes Quality Attribute (QA)s; also, to introduce the information in a proper fashion for users, which includes the required data that supports comparisons between pattern references, and to facilitate their selection processes.

The issues, gaps, limitations, inconsistencies, and conflicts within current SA, QAs, and SPs discovered by this study, such as their poor description and the ignorance of them by developers during software development, has led to important recommendations, as well as suggestions for future research.

The required information from different sectors (government, academia and industry) regarding SPs, SA, SAE, and modelling languages, has been gathered, and analysed through two surveys and a field study.

The strong relationships and influences between the aforementioned areas were introduced
and proven by a case study analysis for the Real-time Control System Real-time Control System (RCS) reference architecture, followed by introducing a conceptual paradigm that aimed to improve and generalise the Moreno et al. [2008] performance model.

The outcomes from this thesis provide the basis for future work. Also, the information from different interdisciplinary knowledge merged to form new concepts for SA evaluation, which are recommended for future study.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special Thanks</td>
</tr>
<tr>
<td>Acknowledgements</td>
</tr>
<tr>
<td>Abstract</td>
</tr>
<tr>
<td>List of Figures</td>
</tr>
<tr>
<td>List of Tables</td>
</tr>
<tr>
<td>Abbreviations</td>
</tr>
<tr>
<td>Preface</td>
</tr>
</tbody>
</table>

## I Introduction

1 Overview

1.1 Introduction ........................... 4
1.2 Initial motivation and research aim ............... 4
1.3 Methodology and Research Design ............... 6
   1.3.1 Research Method .................. 6
   1.3.2 Research Activities ............... 7
   1.3.3 Research Life Cycle ............... 7
1.4 Thesis Scope and Structure ................... 8
   1.4.1 Introduction (PART I) .............. 11
   1.4.2 Contribution (PART II) ............. 11
   1.4.3 Wrapping up (PART III) .......... 12
1.5 Publications ................................ 12
1.6 Summary of Contributions .................... 13

## 2 Background

2.1 Introduction ................................ 16
2.2 Initial research ........................... 18
2.3 Software Architecture Description ............... 27
   2.3.1 Brief Analysis of SA description methods ........ 27
   2.3.2 Views of SA description ............... 33
   2.3.3 Formal methods and languages ............. 35
      2.3.3.1 Common formal methods and languages ........ 37
   2.3.4 ACME in brief .......................... 41
      2.3.4.1 ACME design trade-offs ............... 42
   2.3.5 Conclusion ............................ 43
2.4 Model driven approaches and Architecture ........... 43
   2.4.1 Model driven software development .............. 44
      2.4.1.1 Importance of domains in MDSD ......... 47
   2.4.2 Introduction to MDA .................... 48
      2.4.2.1 MDA Framework ...................... 49
2.4.3 Object Oriented (OO)-method ............... 51
   2.4.3.1 Key aspects of OO-method with respect to SA, SPs, and QAs .......... 52
   2.4.3.2 Mapping OO-Processes to MDA-Processes .......... 53
   2.4.3.3 Extra features of OO-methods missing from MDA .......... 54
   2.4.4 eXecutable and Translatable UML (\textsuperscript{\textregistered} UML) as an example of model automation approach ........ 54
   2.4.5 Key aspects of MDSD approaches with respect to SA, SPs, and QAs .......... 55
      2.4.5.1 Models and transformations ............... 56
II Contribution

3 The dilemma of Software Pattern descriptions with partial solution

3.4 Conflict Example - (Proxy Pattern)

3.6 Proposed solution

3.6.1 Functionality description

3.7 Related work

3.8 Conclusion

4 Factors Influencing Utilisation of Software Patterns: A Questionnaire Analysis Result

4.4 Findings and Recommendations
List of Figures

1.1 Thesis-Structure-Diagram. .................................................. 10
1.2 Dependency relationships between chapters and appendices: Package Diagram. 11
2.1 The main common steps for a Systematic review. ............................ 20
2.2 Comprehensive search and Systematic review processes in IDEFO diagram format as adopted for this study. ....................... 27
2.3 The three main methods for describing SA. ................................... 28
2.4 General prospective of the model types for describing a system. ............ 35
2.5 Evolution of formal methods. .................................................. 36
2.6 Formal methods adoption has grown by 53% within 3 years. ................ 37
2.7 AADL abstraction and key specification elements, after Feiler et al. [2006]. 41
2.8 Main differences between modelling approaches, after Cabor [2014]. ........ 44
2.9 Basic concept of MDA .......................................................... 49
2.10 The basic, extended, and complete MDA framework. ....................... 50
2.11 MDA basic process. ........................................................... 51
2.12 OO – method concept schema. .............................................. 52
2.13 The $\Sigma$ UML concept. ...................................................... 55
2.14 Horizontal and Vertical transformations of models. ........................ 56
2.15 Relationship between models and real world. .................................. 59
2.16 Model to Model Transformation ............................................. 60
2.17 Evaluation techniques. .......................................................... 65
2.18 Activities in SAAM analysis. ............................................... 67
2.19 The four phases of ATAM .................................................... 68
2.20 Steps of ATAM ................................................................. 70
2.21 A conceptual flow of ATAM .................................................. 71
2.22 Steps of ARID ................................................................. 72
2.23 The ALMA method five steps. .............................................. 73
2.24 The ALPSM method six steps. .............................................. 73
2.25 Quality attribute scenario representation model. ............................. 79
2.26 Performance characteristics: stimuli, responses and architecture decisions. 79
2.27 Performance tactics with elaboration ..................................... 82
2.28 Relationships between security attributes, properties, and patterns ........ 83
2.29 Applying Auditability tactic. ............................................... 83
2.30 Similarity between Bass and Barbar tactics. ................................ 86
2.31 Performance framework ..................................................... 89
3.1 Visualizing the Problem Area. .............................................. 109
3.2 The main SPs investigation phases towards producing the solution ........... 109
3.3 Examined references included – (Database snapshot) ......................... 111
3.4 Terminologies of “Pattern” within software development lifecycle .......... 113
3.5 The GoF approach for classifying and describing Proxy pattern, includes all variants and relationships with quality attributes. ......................... 117
3.6 POSA team approach for classifying, describing Proxy pattern, include all variants and relationships with quality attributes. ......................... 118
3.7 Reliability as an example of its QAs sub-categories differences ............... 119
3.8 Reliability sub-categories differences between ISO-9126 and ISO-25010 ... 121
3.9 Overall structure of matrix pages and tabs. .................................... 123
3.10 Landing page, which describe the application and its pages functionalities. . 123
3.11 List of patterns, their other names, and definitions. .......................... 124
3.12 Drop down menu for SPs categorisations. .................................... 125
3.13 GoF categorisation table. .................................................... 125
3.14 Description table for ‘Check Point’ pattern. .................................. 126
3.15 Efficiency (QA) description table, including definitions (SEI, POSA and ISO). Also, the table shows the (SPs) related to Efficiency. ......................... 126
3.16 Snapshot of general matrix that shows the relationships between SPs and QAs; where - $S \Rightarrow Support, H \Rightarrow Hinder, and B \Rightarrow Both$ (conflict). .......................... 128
3.17 Edit existing relation page. ................................................................. 128
3.18 Create new relationships between SP and QA (QA information page- step 2). . 129
3.19 Search relation page, either by pattern or by quality attribute. Also, it could search all conflict relations within the database .................................................. 129

4.1 The survey process that applied for Chapters 4 and 5, after Kasunic [2005] .... 135
4.2 The three sections of the questionnaire. .................................................. 141
4.3 The level of participants’ agreement to the four statements (proposed solutions). 149
4.4 Mean response of how often developers’ uses software styles/patterns during their work by their general field of expertise. ................................. 150
4.5 Mean plots (a) to (d) for each of the statements (Q17 to Q20) in Y-axis, along with developers’ work sectors in X-axis. (A=Academia; I=Industry; G=Government). .................................................. 153
4.6 The mean of the items for developers who are aware of software styles/patterns. 154
4.7 Developers’ work sectors vs how often they used software patterns (frequency in stacked bar chart). .................................................. 156

5.1 The two pillars to investigate SA, SAE, and SPs relationships. .................... 164
5.2 The three main sections of the survey. .................................................. 166
5.3 The main five SA elements influencing SAE. ......................................... 166
5.4 The three main sections and their sub-sections of the analysis. ................. 172
5.5 Respondent’s general field of expertise regarding software development. .... 172
5.6 The main factors Encouraging the utilisation of modelling techniques to describe SA. .................................................. 177
5.7 The main factors Discouraging the utilisation of modelling techniques to describe SA. .................................................. 178
5.8 The percentage of respondents who are aware of any system/software architectural tactics/metrics that have been or are being used for evaluating architecture description models. .................................................. 181
5.9 Respondents who identified the most important factors that could SUPPORT SA quantitative evaluation. .................................................. 181
5.10 Means for the statements in (Q15-Q19), with error bar and assumed mean line. 183
5.11 Means for the statements in (Q20 to Q23), with error bar and assumed mean line. 184
5.12 The Field study phases. ................................................................. 201

6.1 RCS architecture analysis steps. .......................................................... 217
6.2 RCS overall architecture (for Military Application), after Finkelstein [2008] .... 218
6.3 Functional relationships between modules of Elementary Loop of Functioning (ELF), after Meystellen et al. [2002]. .................................................. 218
6.4 RCS computational node Inner-structure, after Finkelstein [2008]. ............ 221
6.5 A top-level Elementary Loop of Functioning, with two control levels (nodes), after Meystellen et al. [2002]. .................................................. 222
6.6 Relationships within a typical node of RCS Architecture, after Meystellen et al. [2002]. ................................................................. 223
6.7 The link between the entire RCS appraisal process and Chapter 3 findings .... 231
6.8 Model driven engineering and Model Driven Analysis. ............................ 232
6.9 Conceptual patterns evaluation schema. .................................................. 233
6.10 The forces within Pattern ingredients, after Tešanovic [2005]. .................... 233
6.11 Derived evaluation metrics starting from the right tactics. .......................... 234
6.12 Brief of the scenario-based dependencies and activities during SAE, after Kazman et al. [1996]. ................................................................. 235
6.13 Pipe and Filter family illustrated through ACME-Studio tool. .................... 235
6.14 Subset of the security pattern language. .................................................. 239
6.15 Example of the model processes for (coupling measure), by utilising SysML and Artisan-tool. ................................................................. 240

B.1 MDA advantages, After Flint [2008]. ..................................................... 301
B.2 MDA limitations, After Flint [2008]. ......................................................... 302

C.1 Create new relation between SP and QA, Pattern information page. .......... 311
C.2 Layers pattern description table, including definition, and related (QAs). .......... 312
C.3 Overview page ................................................................................. 312

D.1 Stacked bar chart: Agreement with the four statements by different sectors. .... 319
D.2 Stacked bar chart: Years of experience in software development field with the developers awareness of software style/patterns. ................................. 321
D.3 Overview page - for the primitive analysis database .................................... 324
D.4 Responses primitive analysis page - Respondent’s countries, total answers, percentage of their answers out of 20 questions. ........................................ 324
D.5 Selection criteria page - were one or more questions should be selected and submitted by clicking ’submit’ button to get the analysis results. .................. 325

E.1 Bar chart for the best language to use to describe software/system architecture as identified by the respondents. ............................................. 332
E.2 Box plot of two items Q9 and Q10............................................................... 332
E.3 Pie Chart of the percentage of the respondents who know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics. ......................................................... 333

F.1 Each layer of the system contains a number of RCS-nodes, each containing – Behaviour Generation (BG), World Model (WM), Sensory Processing (SP), and Value Judgement (VJ) - modules. The nodes are interconnected as a layer style, or lattice, through the communication system, Meystel et al. [2002]. ............... 390
F.2 Hierarchical image and entity frames within Sensory Processing (SP) component, After Meystel et al. [2002]. ............................................................ 391
F.3 Snapshot for the RCS Node structure with SysML, using Artisan tool. ........... 392
F.4 Extracted RCS architecture for coupling evaluation .................................... 393
F.5 Modified model from the extracted RCS architecture, for comparison purpose. .. 393

G.1 System Architecture Evaluation (SysAE-Product) structure, Using component base products approach ......................................................... 396
G.2 Requirement diagram – SAQEF. Requirement Hierarchy. ............................... 399
G.3 SAE – Operational concept (Graphical representation). ................................ 400
G.4 SAE – Operational concept (Model components representation). .................. 400
G.5 UPDM structure. ............................................................................ 401
G.6 UPDM structure (package diagram) - Artisan tool. ..................................... 402
G.7 SAQEF Packaging structure –(using packages and views) - (Package diagram) – Artisan tool – The package ’SAQEF Model’ illustrate how the model organised into packages that contain model elements .............................................. 404
G.8 SAQEF (Efficiency assessment) – More details with employment of DoDAF views - (Requirement diagram) – Artisan tool ........................................... 405
G.9 SAQEF-AV-1, used of DoDAF views - All view (AV-1) - Artisan tool ............ 406
G.10 Measurement (Dimensions and Units) – SAQEF profile – Artisan tool .......... 408
G.11 Constraints and parameters details for (System coupling measurement) in Block Definition Diagram format (bdd) - SAQEF profile - Artisan tool. ............... 409
G.12 Parametric Diagram – System coupling. The constraint block from Figure G.11 has been expanded in this Figure where its equation is the constraints property (yellow box), which is belong to its block. Constraint notes are used to explain the system coupling equation and expected output limits - SAQEF profile - Artisan tool.
# List of Tables

2.1 This is the review strategy that has been applied in Chapter 2, followed by Chapters 3, 4 and 5). ................................. 21
2.2 Formal, Semi-formal, and Informal ways of representing a section of Pipe-Filter family. ........................................... 34
2.3 Rushby’s four levels of formalisation. ................................. 38
2.4 General comparisons between ACME and AADL main components. ................................................... 40
2.5 Advantages of MDSD ................................................... 45
2.6 Mapping (Rushby 1993b) formality levels into the model maturity levels by Rensink et al. [2006]. ......................... 47
2.7 Analogies between MDA and the OO-method. ......................... 54
2.8 MDSD approaches, Transformations Methods, and QAs – Comparison. .................................................. 58
2.9 Comparisons between different evaluation techniques. ..................... 76
2.10 Absence of solutions to some problems in Babar et al. [2005] model. ...... 84
2.11 Non-mapped elements between Babar et al. [2005] model and SEI model. . . . 84
2.12 Software quality characterisation. ....................................... 95
2.13 Set of quality attributes trade-offs. ..................................... 97
3.1 Descriptions of the 7-Analysis Steps for the Targeted Resources. .......... 110
3.2 Example of extracting pattern data in a format suitable to assist evaluation. 127
4.1 Summary of the questions and the objectives of each section ................. 135
4.2 Section 1. Background Questions (Questions marked with (*) are mandatory). 136
4.3 Section 2. Pattern utilisation questions. ...................................... 137
4.4 Section 3. Documentation of Software Patterns. ............................ 138
4.5 Geographical distribution of the survey respondents by country ........... 140
4.6 Two dimensional matrices analysis (only two questions will be analysed together). ........................................... 142
4.7 Three dimensional matrices analysis (only three questions will be analysed together). ............................................ 142
4.8 Four dimensional matrices analysis (only three questions will be analysed together). ............................................. 142
4.9 Frequency distribution of the questions regarding personal expertise and Chi-square test result for equality of group frequencies. ...................... 145
4.10 Frequency distribution summary for (Q6 to Q16). .......................... 147
4.11 Independent sample t-test for equality of two population group means of “how often developers used software styles/patterns during their work” by developer’s field of expertise. ....................... 151
4.12 Independent sample t-test for equality of two population group means of “how often developers used software styles/patterns during their work”. .... 155
4.13 Cross tabulation of three work sectors with software styles/patterns usage and consideration of quality attributes and Chi-square test of independence between attributes. ........................................ 157
4.14 Summary of the main significant analysis results with associated tables (for better traceability) .................................. 160
5.1 Section 1. Background Questions (Questions marked with (*) are mandatory). 166
5.2 Section 2. Software architecture utilisation questions. ........................ 167
5.3 Section 3. SA evaluation methods and tools. ................................... 169
5.4 Two dimensional matrices analysis (only two questions will be analysed together). ............................................ 174
5.5 Frequency distribution of single response categorical questions/items and Chi-square test results for equality of group frequencies; (Q2-Q5, Q8, Q11, Q12). 175
5.6 Descriptive statistics for Likert scaled items along with one-sample t-test results for testing assumed mean of 3 (Neutral value), (Q9, Q10, Q15-Q23). 179
5.7 Cross tabulation between respondents’ experiences and the main factors that DISCOURAGE the utilisation of modelling techniques to describe SA, (Q3 with Q7). 186
5.8 Frequency distribution of the multiple response questions/items, (Q6, Q7, Q13, Q14). 189
5.9 Pearson Chi-square test results between questions Q5 and Q6. 191
5.10 Pearson Chi-square test results between questions Q5 and Q7. 192
5.11 Independent samples t-test results between both statements for Equality of Grouped Population Means corresponding to the categories (Yes or No). 193
5.12 Pearson Chi-square test results for analyses of questions Q10 and Q13. 194
5.13 Summary of the main analysis results with associated tables (for better traceability). 197
5.14 The Organisations Teams – preliminary phase 203
5.15 General criteria for selecting field study environment. 204
5.16 Summary of the Preliminary Visit Findings. 204
5.17 Investigated Projects during the case study. 207
5.18 Field study findings. 208
6.1 BCK styles and QAs within RCS architecture. 219
6.2 Mellor [2009] styles and QAs within RCS architecture. 224
6.3 QAs trade-offs within RCS Styles. 230
6.4 Reliability/availability evaluation, methods, data, and the generic framework. 238
B.1 ADL design goals 304
B.2 ADL characteristics 304
B.3 The Most Known ADLs that still supported: the table contains also approaches which are considered non-conventional ADLs, since they might neglecting fundamental aspects, After Rech et al. 2009, pp 267 306
C.1 Database application revisions. 313
D.1 Two dimensional matrices analysis methods. 315
D.3 Cross tabulation of three work sectors with software styles/patterns usage and consideration of quality attributes and Chi-square test of independence between attributes. 317
D.2 Descriptive statistics and comparison of the population mean to a hypothesized neutral value of (3), using t-tests for each of the statement with the general field of expertise regarding software development. 318
D.4 Descriptive statistics and test of the population mean to a hypothesized neutral value at (3),using t-test for each of the statements with the sectors of work experience. 320
D.5 Percentage frequency table: Awareness of software style/patterns by developers. 321
D.6 ANOVA-1, Analysis of Question 1/ Option 1 “Requirements elicitation/modelling/analysis”, for Q17 to Q20 by Q2. 322
D.7 Cross tabulation Analyses for (Q2 and Q3 with Q4 and Q5). 323
D.8 Cross tabulation and test of independence between how often developers used software style/patterns and whether the developers support standard documentation practices for software patterns. 325
D.9 Cross tabulation of how often developers used software style/patterns with the main factors that discourage and encourage the utilisation of software patterns by developers and Chi-square test of independence between attributes. 326
E.17 ANOVA procedure for testing equality of group means for the selected item corresponding to the categories of how often respondents used models to describe software/system architecture during their work.

349

E.18 Multiple comparison test results using LSD method.

349

E.19 ANOVA procedure for testing equality of group means for the selected item corresponding to the categories of how often respondents used models to describe software/system architecture during their work.

351

E.20 Multiple comparison test results using LSD method.

351

E.21 Independent sample t-test results for equality of two population group means of Likert scaled items Q9, Q10 and Q15 to Q23 by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture (Q6). Groups formed by the developers who agreed with a Q6 item (Q6CB1 to Q6CB8) and developers who didn’t agree with that item.

353

E.22 Independent sample t-test results for equality of two population group means of Likert scaled items Q9, Q10 and Q15 to Q23 by the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture (Q7). Groups formed by the developers who agreed with a Q7 item (Q7CB1 to Q7CB5) and developers who didn’t agree with that item.

354

E.23 Pearson Chi-square test results for analyses of questions Q10 and Q17.

354

E.24 Cross tabulation of “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “Most of the existing software architecture evaluation methods, produce qualitative results”.

355

E.25 Cross tabulation of the most important factors that could SUPPORT quantitative evaluation for any SA and the opinion about the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”.

355

E.26 Pearson Chi-square test results for analyses of questions Q13 and Q22.

356

E.27 Pearson Chi-square test results for analyses of questions Q10 and Q13.

356

E.28 Cross tabulation of respondent’s sectors in which respondents gained most of their general software development experience and the main factors that encourage the utilisation of modelling techniques to describe software/system architecture.

356

E.29 Pearson Chi-square test results for analyses of questions Q2 and Q6.

357

E.30 Cross tabulation of respondent’s sectors in which respondents gained most of their general software development experience and the main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture.

358

E.31 Pearson Chi-square test results for analyses of questions Q2 and Q7.

359

E.32 Pearson Chi-square test results for analyses of questions Q2 and Q13.

359

E.33 Cross tabulation of sectors in which respondents gained most of their general software development experience and the most important factors that could SUPPORT quantitative evaluation for any SA.

359

E.34 Cross tabulation of sectors in which respondents gained most of their general software development experience and the most important factors that could HINDER quantitative evaluation for any Software Architecture (SA).

360

E.35 Pearson Chi-square test results for analyses of questions Q2 and Q14.

360

E.36 Pearson Chi-Square test results for testing association between years of experiences of respondents in the software/systems development field and how often the respondents use models to describe software/system architecture during their work when the general field of expertise regarding software development is “Requirements elicitation/modelling/analysis”.

361
E.37 Cross tabulation of years of experiences of respondents in the software/systems
development field and how often the respondents use models to describe software/system architecture during their work when the general field of expertise regarding software development is “Requirements elicitation/modelling/analysis”.

E.38 Multiple Comparisons between Q2, Q22, when Q1 = Project Management

E.39 One way ANOVA table to test the difference among group means for Q22, Q2, when Q1 = Project Management

E.40 Cross tabulation Analyses for (Q2 and Q3 with Q4 and Q5).

E.41 Cross tabulation of sectors in which respondents gained most of their general software development experience and whether respondents were aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models (e.g. detecting attacks for security).

E.42 Pearson Chi-square test results for analyses of questions Q2 and Q11.

E.43 Pearson Chi-square test results for analyses of questions Q3 and Q5.

E.44 Cross tabulation of respondent’s years of experience in total in the software/systems development field and how often they used models to describe software/system architecture during your work.

E.45 Pearson Chi-square test results for analyses of questions Q3 and Q7.

E.46 Independent samples t-test results for the selected items for equality of grouped population means.

E.47 Cross tabulation of the developer’s years of experience in the software/system development field against selected categorical questions and corresponding results for \( \chi^2 \)-test of independence.

E.48 Cross tabulation between the developer’s awareness regarding modelling languages and selected categorical questions.

E.49 Cross tabulation of how often developer’s used models to describe software/system architecture during their work against selected categorical questions and corresponding results for \( \chi^2 \) test of independence.

E.50 Cross tabulation of the developer’s general field of expertise regarding software development against selected categorical questions and corresponding results for \( \chi^2 \) test of independence.

E.51 Independent sample t-test for equality of two population group means of “developing software/system architecture using current architectural frameworks increases the reliability, standardisation, and reusability of the resulting architecture” by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture.

E.52 Independent sample t-test for equality of two population group means of “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture.

E.53 Summary table of \( \chi^2 \)-test results for testing pairwise independence between categorical variables.

E.54 Analysis of variance (ANOVA) results for Q9, Q10 and Q15 to Q23 by the single response categorical variables.

E.55 Summary table of Chi-Square test results for testing pairwise independence between two Likert scale variables, where (the row and column variables are Likert scaled).
E.56 Summary table of Chi-Square test results for testing pairwise independence between a categorical variable and a Likert scale variable (row variables are categorical and column variables are Likert scaled). . . . . . . . . . . . . . . 380
E.57 Checklist questions, Organisation answers and feedback for: Site1 . . . . . . 381
E.58 Checklist questions, Organisation answers and feedback for: Site2 . . . . . . 383
E.59 Checklist questions, Organisation answers and feedback for: Site3 . . . . . . 384
Abbreviations

Symbols

$\UML$ EXecute and Translatable UML

A

AADL Architecture Analysis & Design Language

ABAS Attribute-Based Architectural Styles

ACO Airspace Control Order

ADL Architecture Description Language

ADR Active Design Reviews

AHP Analytical Hierarchy Process

AI Artificial Intelligence

AIS Artificial Intelligence System

ALMA Architecture Level Modifiability Analysis

ALPSM Architecture Level Prediction of Software Maintenance

ANOVA Analysis of variance

ANU Australian National University

AOC Air Operations Centre

ARID Active Reviews for Intermediate Design

ATAM Architecture Tradeoff Analysis Method

ATO Air Tasking Order

B

BCK Bass, Clement, and Kazman

BG Behaviour Generation

C

$C^3$ Command, Control, and Communication

$C^4I$ Command, Control, Communications, Computers, and Intelligence

CBO Coupling Between Objects
CC  Component-and-Connector
CDG  Component Dependency Graph
CMMI  Capability Maturity Model Integration
COSMIC  Common Software Measurement International Consortium
COTS  Commercial off-the-shelf
CPU  Central Processing Unit
CSM  The Core Scenario Model
CSP  Communication Sequential Processes

D
DB  Database
DoD  Department of Defence
DoDAF  Department of Defence Architecture Framework
DSL  Domain Specific Language

E
EMF  Eclipse Modelling Framework
ERP  Enterprise Resource Planning
ESAAMI  SAAM by Integration in the domain
EX  Executer

F
FSM  Finite State Machine

G
GoF  Gang of Four
GUI  Graphical User Interface

I
ICM  Intermediate Constructive Model
ICT  Information and Communication Technology
IDF0  Integrated DEFinition language 0
IDL  Interface Description Languages
IEC  International Electrotechnical Commission
IEEE  Institute of Electrical and Electronics Engineers
IIS  Internet Information Service
IM  Implementation Model
ISO  International Organisation for Standardisation
IT  Information Technology

J
JA  Job Assignor

K
KD  Knowledge Database

L
LSD  Least Significant Difference
LTM  Long Term Memory

M
MAAADL  MontiArc Automaton Architecture description language
MDA  Model Driven Architecture
MDD  Model Driven Development
MDE  Modelling Development Engineering
MDSD  Model Driven Software Development
MDT  Model Development Tools
MODA  Ministry of Defence and Aviation
MoDAF  Ministry of Defence Architecture Framework
MVC  Model View Controller

N
NIST  National Institute of Standards and Technology
NML  Neutral Messaging Language
NOC  Number of Components

O
OASIS  Administrative Systems-development in Incremental Steps
OCL  Object Constraint Language
OMG  Object Management Group
OO  Object Oriented
OOD  Object Oriented Development
OODA  Observe, Orient, Decide and Act
ORM  Object-Role Modelling

P
PAC  Presentation-Abstraction-Control
PCM  Palladio Component Model
PDP  Policy Definition Point
PEP  Policy Enforcement Point
PIM  Platform-independent model
PL  Planner
POSA  Pattern-Oriented Software Architectures
PSM  Platform-specific model

Q
QA  Quality Attribute
QML  Quality of Service Modelling Language

R
RAM  Random Access Memory
RCS  Real-time Control System
REST  Representational State Transfer
RMA  Rate Monotonic Analysis
RSADF  Royal Saudi Air Defence Force
RUP  Rational Unified Process

S
SA  Software Architecture
SAAM  Software Architecture Analysis Method
SAAMCS  SAAM founded on Complex Scenarios
SAAMER  SAAM Evolution and Reusability
SACM  Saudi Arabian Cultural Mission
SADL  Semantic Application Design Language
SAE  Software Architecture Evaluation
SAQEF  Software Architecture Quantitative Evaluation Framework
SC  Scheduler
SD  Standard Deviation
SEI  Software Engineering Institute
SNA  Survivable Network Analysis
SOMF  Service-Oriented Modelling Framework
SoS  System-of-Systems
SP  Software Pattern
SPE  Software Performance Engineering
SPr  Sensory Processing
SPSS  Statistical Package for the Social Sciences
SQL  Structured Query Language
STM  Short Term Memory
SysAE  System Architecture Evaluation
SysML  System Modelling Language
T
TPR  Thesis Proposal Review
U
UGV  Unmanned Ground Vehicle
UI  User Interface
UML  Unified Modelling Language
UPDM  The Unified Profile for DoDAF/MoDAF
V

**VDM** Vienna Development Method

**VJ** Value Judgement

W

**WM** World Model

X

**xADL** A Highly Extensible Architecture Description Language

Z

**Z** Z notation, named after Zermelo–Fraenkel set theory
Preface

After 12 years of experience in software and systems engineering development, as an officer in the military, this research has been initiated. Its main aim is to ensure that I come up with solutions or new findings, in order to make valuable improvements and contributions to the software and systems engineering domains, and to enhance the ‘Information Technology (IT)’ development process within my sponsor’s projects, the Ministry of Defence and Aviation (MODA).

My journey in regard to this research started two years before I registered for a PhD program at the ‘Australian National University (ANU)’, during my masters degrees at the same university, while I was learning new concepts, methods, and processes concerning SA domains. I also participated in the development processes of several projects and observed how SA was treated. I later revisited the causes for those projects’ successes and failures. An example is an integration project for different weapons (worth 96 million US-dollars), which was extended from 1 year-to-6 years and changed from a Fixed-Price contract into a Cost-Plus contract. I believe one of the reasons that cause such failures is ignorance of its architecture development.

Therefore, when I started this research, I was determined to explore practices that could be applied to software-intensive projects in general, and its architecture in particular, to enable them to meet the requirements of different stakeholders.

I started by first holding the proposition that challenges possessed by the architecture are important key factors for project failure. Based on this proposition, I was able to dig deeply into the roots of SA, and SAE by going through various literature from different scholars. Particularly, I examined the description, modelling, documentation, evaluation methods, and QAs of SPs, SA.

Following this, I was able to come to a conclusion that improving (SPs, SA, and SAE) methodologies, descriptions, processes, and documentation should improve software and systems engineering practices. In general, the development of software or systems engineering approaches are interdisciplinary in nature.

Thus, I performed an extensive study of SPs, how SA is currently described, modelled, utilised, and evaluated, as well as SAE methods and utilisation, within the current literature, and organisational development projects (small to large).

My inquisitiveness and investigation regarding the aforementioned issues made me realise that improvements in industry practices can be achieved through creating and utilising software and systems architecture in a proper fashion. These enhancements should benefit the overall outcome, as well as capability in the development of IT systems in general.

Moreover, the comments that I have received regarding my first thesis submission were extremely valuable to me, because they gave me more time to closely perceive and participate in current projects. In particular, I was able to interact and work with several ‘professional groups’ from different disciplines, with the goal of understanding the wider impact of SA on
existing (developed), and under development systems.

It is therefore apparent that SA is an important phase of the software development lifecycle. As a result, this study has discovered new findings, proposed new solutions, and enhanced existing performance models, which are reported here. Also, this work helps to set the roadmap for future research, by proposing new conceptual thoughts named ‘SysAE and SAQEF’, which have emerged as a result of this overall effort.

Hassan Almari
Canberra, Australia
Introduction
Chapter 1

Overview

Slowly, at every level, the arrangement of wholes becomes so dense that there are no gaps between the wholes: every part, and every part between two parts, is whole.

*Alexander* [1979, p490]
CHAPTER 1. OVERVIEW

1.1 Introduction

The intention of this research is to improve Software Architecture (SA) descriptions and Software Architecture Evaluation (SAE) by exposing issues in their current approaches. Fulfilling the intention includes disambiguation of conflicting definitions and opinions concerning the documentation of Software Pattern (SP)s, their relationships with Quality Attribute (QA)s, and their effectiveness in determining SAs and their evaluation.

In this introductory chapter, I present an overview of the study that includes information to assist the reader to navigate this dissertation. I conclude this chapter with a summary of work published during this study and contributions made to engineering research and practice.

1.2 Initial motivation and research aim

In conducting the research reported in this thesis, an initial motivation was present already in the form of the following three interrelated statements:

Firstly, “If we are to agree on what it means to document a software architecture, we should establish a common basis for what it is we’re documenting”, Bachmann et al. [2011, p3]. Secondly, “Architecture assessments are essential for avoiding, identifying, or mitigating risks”, Mistrik et al. [2014, p11]. Thirdly, “Using a pattern or style means making successive design decisions that eventually result in an architecture”, Bachmann et al. [2011, p35].

The foundational theory behind this work (as expressed by many others including Qin et al. [2008], Bachmann et al. [2011], and Bass et al. [2013]) is that SA, SAE, and SPs, are strongly related. Whilst the theory might hold, the various artefacts and practices that have been developed over time (including their documentation processes) do not, which prevents the maximum utilisation of many of the existing achievements within SPs, SA, and SAE domains.

This (current) situation has led to an initial aim of finding ways to realize current issues regarding SA descriptions, SAE methods, and documentation of software patterns/styles\(^1\), in order to identify areas of improvement.

Preliminary research (reported in Chapter 2) shows that SA and software Styles/Patterns have a strong relationship and improving one will improve the other. It also explains that SAE is affected by several factors, such as SA description languages including SPs, the level of formality, documentation techniques, standardisation, and selected evaluation methods. After all: “Architectural evaluation of a software/system is crucial for its success”, Reussner et al. [2005].

In order to assess the architectural fitness of software systems, a number of evaluation methods have been proposed, such as Architecture Tradeoff Analysis Method (ATAM) and Software

\(^1\)Different books distinguish between styles and patterns, where styles are considered for architecture and patterns are considered for design. In this thesis, the difference between style and pattern is not critical, both are considered as a recurring solution to the same problem in different contexts.
Architecture Analysis Method (SAAM), and Active Design Reviews (ADR). Most of the existing methods are intended for evaluation of a single architecture at a certain point in time.

Furthermore, the results from using MANY/SOME methods are highly dependent on the person performing the evaluation and generally cannot be used to compare different architectures. Most of the current mature architecture evaluation methods, such as ATAM and SAAM use qualitative techniques that are typically applied through the use of scenarios. Consequently, the interpreted results depend heavily on the choice of scenarios to evaluate certain QA. The generation of these scenarios is solely based on the vision and requirements of the stakeholder(s). The conflicts between stakeholders’ requirements raises a major challenge to software architects. This was noticed by Avison et al. [1999] and Baskerville et al. [2004] during their studies for ATAM and SAAM methods. The same challenge is confirmed by Qin et al. [2008, p227-228], Bass et al. [2013, p401], and Shreelekhya et al. [2016].

Moreover, SAAM has at least one pitfall in that it does not provide clear quality metrics for the architectural characteristics that are being analysed, Clements et al. [2002a]. In addition, the assessment of some properties requires experts. For example, evaluation of security is not usually easy, due to the lack of adequately qualified resources. Another major problem is that most of the evaluation methods are ad-hoc processes or minimally-automated and thus they are prone to error. Additionally, most of these methods, especially the semi-automated ones, are developed to evaluate one quality attribute, such as in the work done by Moreno et al. [2008] to analyse performance.

Further research, as reported in Chapter 2, shows that the SA qualitative evaluation methods received much more attention than quantitative evaluation methods, which renders the latter methods as more limited and in need of more research, and exhibiting less maturity.

In concluding this preliminary research, the necessity to go further in order to investigate SPs documentation and utilization, SA, and SAE methods, it was important to understand and discover the factors influencing SAE approaches.

Hence, based upon the preliminary study, questions have been formulated (as below) so as to promote understanding of the SA description and evaluation, and to develop a viable solution:

1. What effect do description languages, standardisation, evaluation methods, modelling techniques and their documentation have on ‘SPs and SA’ utilisation and evaluation?
2. How can ‘SPs and SA’ utilisation and evaluation be improved, including minimising the effects of any hindrances?

Answering these questions is important, since “An unsuitable architecture will precipitate disaster on a project”, Clements et al. [2002a].
Thus, the aim of this research is to investigate the SA and SAE domains through literature review, questionnaires, field study, and analyses. As a result, issues and/or links between different disciplines, such as SPs, modelling languages, and SA description and evaluation techniques, were presented, in order to resolve identified issues, bridge the gaps, and overcome the limitations of SAE current methods.

1.2.1 Contributing Factors to the Problem Domain of this Thesis

The following points list the main factors that make this research worthwhile:

1. Lack of architectural evaluation approaches that apply standard languages and that are fully automated.
2. The absence of well-known, wide-spread standardized languages and tools in current architectural evaluation studies.
3. There are no/few robust architectural evaluation models that can help architects and developers to check their architecture against intended and required ‘qualities’ with no need for experts.
4. There is no existence of a facility (such as a database, repository, or thesaurus etc.) that demonstrates the relationships between SPs and QAs based on reliable references.
5. There seems to be a dearth of experts on evaluation methods at an architectural level.

1.3 Methodology and Research Design

In this section, the ‘best fit’ research methods, activities, and life cycle are described, according to aims and context.

1.3.1 Research Method

Software engineering research methods were classified by Adrion [1993], as follows:

1. “Scientific Method: observe the world; propose a model or theory of behaviour; measure and analyse; validate hypothesis of the model or theory; and if possible repeat.
2. Engineering Method (evolutionary paradigm): observe existing solutions; propose better solutions; build or develop; measure and analyse; then repeat until no further improvements are possible.
3. Empirical Method (revolutionary paradigm): propose a model; develop statistical or other methods; apply to case studies; measure and analyse; validate the model; then repeat.
4. Analytical Method: propose a formal theory or set of axioms; develop a theory; derive results; and if possible compare with empirical observations”
1.3. METHODOLOGY AND RESEARCH DESIGN

The Engineering Method is the best fit for achieving the thesis aim. In fact, the research started by observing the existing approaches that describe and evaluate SA, SAE methods, and SPs, using several observation methods, such as analysing the current solutions, surveys, and field study. Demonstration of the relationship between SPs and QAs was achieved with the development of a database that contains the required information. A critique of a reference architectural model has been carried out in Chapter 6 in order to provide evidence of the strong relationship between SPs, SA, and QAs. The engineering method is the best approximation to the research activities applied in this thesis, and will continue to be used during the compilation of recommended future work until no further improvements are possible. The latter will hopefully be fulfilled via the link between this thesis aim and suggested future work as its extension.

1.3.2 Research Activities

"Research activities are sets of tasks that may be carried out to implement part of a research approach", Flint [2006].

According to a proposed model by Glass [1995], the four general computing research phases are:

1. "The Informational Phase: gathering or aggregating information via reflection, literature survey, people/organisational survey, or poll (e.g. Delphi approaches).
2. The Propositional Phase: proposing and/or formulating a hypothesis, method or algorithm, model, theory, or solution.
3. The Analytical Phase: analysing and exploring a proposition, leading to a demonstration and/or formulation of a principle or theory.
4. The Evaluative Phase: evaluating a proposition or analytic finding by means of experimentation (controlled) or observation (uncontrolled, such as a case study or protocol analysis), perhaps leading to a substantiated model, principle, or theory".

Each one of these phases could have one or many activities. This thesis includes all the activities above as explained in Section 1.4.

The adaptation and sequencing of Glass [1995] model is varies from research to research, based on the need and the research life cycles, Flint [2006].

Important point has been considered during the research activities:

- Confidentiality of the data and results
  The field study included in Chapter 5 and associated appendix, involved military sites, and hence there shall be the need for retaining some sensitive information.

1.3.3 Research Life Cycle

"A life cycle orientation that addresses all phases", Blanchard et al. [1990].
Every project should have certain phases that form a cycle, in order to plan, develop, manage, and evaluate a successful outcome, which is known as the project life cycle, *Kerzner* [2013].

In order to successfully perform the research activities described in Section 1.3.2, using the research method explained in Section 1.3.1, the life cycle of the research becoming more clearer and understandable.

There are several life cycle approaches that explain different ways in which researchers can perform their research; however listing and explaining these methods are out the scope of this thesis. However, *Flint* [2006] explains some of these approaches in his thesis, especially the ones related to software engineering domain.

In this research, I employed the first phase of the ‘Research-then-Transfer’ life cycle, described by *Potts* [1993]. This approach starts with motivation and initial objectives to utilise a method or technology to resolve an existing industrial problem, then the research continues with slight or no involvement with industry. When the research is considered ‘ready for transfer’, it is presented to industry, *Flint* [2006]. This research problem related to industry and academic fields. Thus, there is slight involvement with industry organisations in different aspects of the research through the surveys and the field study. This research concept is a continuous effort that could be evolved even after the thesis is done. So, when the framework and models that are proposed by this study are fully developed, then they will be transferred to the industry.

### 1.4 Thesis Scope and Structure

The scope of this research lies within the context of software engineering to improve SA and its evaluation techniques. While the current deficiencies in SA descriptions and SAE mechanisms are discussed, the development and evaluation of a completely new SAE framework within a more general context is beyond the scope of this thesis.

Using the operational view of the Department of Defence Architecture Framework (DoDAF), the structure of this thesis is depicted in a System Modelling Language (SysML) activity diagram; Object Management Group (OMG), *Weikens* [2007, p257–264], as shown in Figure 1.1. The vertical swim-lanes (partitions) of the activity diagram in Figure 1.1, show how this thesis has been organised into three parts, each part comprising two or more chapters. The rounded boxes are the research activities that represent the chapters of this thesis, and the directional arrows between the boxes represent the flow of ideas, inputs, and outputs. The commented green boxes represent key contributions made by the research and the output of each activity.

The dependency relationships between chapters and appendices are illustrated in the package diagram in Figure 1.2.

Through reading the Preface, the reader can gain a concise overview; however, further summary details can be gained by reading the introduction to each chapter of (PART II).
1.4. THESIS SCOPE AND STRUCTURE

<table>
<thead>
<tr>
<th>Typographical and other conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within the structure of this Thesis, there are several emphasising styles being used as follows:</td>
</tr>
<tr>
<td>• Each key concept is described in a separate section.</td>
</tr>
<tr>
<td>• Within the text of this Thesis, the names of key concepts, important words, phrases, or sentences are printed in italic font and/or bold font.</td>
</tr>
<tr>
<td>• Some of the most important paragraphs include comments, findings, summaries of sections, and key concepts are placed on a grey box such as this.</td>
</tr>
<tr>
<td>• Figure and table numbers that start a with ‘letter’ are a reference to their placement in an Appendix of the same letter.</td>
</tr>
</tbody>
</table>
Figure 1.1: Thesis-Structure-Diagram.
1.4. THESIS SCOPE AND STRUCTURE

Figure 1.2: Dependency relationships between chapters and appendices: Package Diagram.

1.4.1 Introduction (PART I)

The first part (PART I) of the thesis comprises two chapters and two associated appendices (A and B):

In Chapter 1, I outline the motivation, aims, scope, and structure of the research.

A literature review is reported and discussed in Chapter 2. The findings of the analysis are reported in three main areas: i) SA descriptions and formality level, ii) model driven methods of SA in architectural context, and iii) SAE. Also, this preliminary research explores the potential “forces” that have an effect on software style/pattern descriptions, modelling techniques, and standardisation. A comparison between existing software architecture evaluation methods and categorizations of quality characteristics has been investigated, analysed and reported. This background study of the current state-of-the-art highlighted challenges and gaps within SA and SAE.

1.4.2 Contribution (PART II)

The second part (PART II) of this thesis comprises four chapters, Chapter 3 to Chapter 6.

In Chapter 3, and associated Appendix C, I present an in-depth investigation and analysis of
six reliable software pattern resources in the context of their relations to quality characteristics. The study findings are stored in a database. Issues raised from the analysis are illustrated and discussed.

The results of a questionnaire-based survey are in Chapter 4 and associated Appendix D. These relate to tSPs utilisation factors. The survey was undertaken in order to support and enhance, or otherwise, an understanding of the value of Chapter 3 outcomes, and to identify any issues related to the usability of SPs amongst software engineers and developers.

Illustration of the challenging factors that have an effect on software architecture modelling and evaluation techniques are captured through a second survey questionnaire and a field study, with the results reported in Chapter 5 and associated Appendix E.

Both questionnaires in Chapter 4 and 5 were distributed to software developers within six countries. The degree of experience in software engineering of each participant varied significantly, but no participant had less than 5 years experience. Whereas, the field study was conducted in a military site during the development of many large-scale systems by several international companies, which means that these large scale Information Technology (IT) projects joined highly experienced people from both industrial and military domains.

In Chapter 6 and associated Appendix F, I provide a case study through an exploration and critique of an architecture reference model called “Real-time Control System (RCS)”. In this chapter, I demonstrate the robust relations between SA, SPs, and QAs. I also explain how they are bound together and affect each other through a well-known architecture utilised in industry. Furthermore, this chapter introduces an evaluation concept for SA, including an example to explain how the model works.

1.4.3 Wrapping up (PART III)

The third part (PART III) of the thesis is formed by one chapter, Chapter 7 titled “Discussion and Conclusion”. In Chapter 7 and associated Appendix G, I present a summary of the research work and its limitations, together with my conclusion and suggestions for further research.

In the future work section, I propose two initial concepts: first is System Architecture Evaluation – SysAE profile, to evaluate both hardware and software architectures. The second concept is an evaluation framework and model named SAQEF, for evaluating software architecture quantitatively. Although the rationale and justification of both concepts are briefly described in Appendix G, I have included these two concepts in the future work section, as they need more research and development to fully prove their applicability in the problem domain.

1.5 Publications

The results of the preliminary research presented in Chapter 2, and published in the following refereed conference papers form the basis of the contribution made by this thesis.
1.6 Summary of Contributions

There are five key contributions arising from the research undertaken.

1. **SPs and QAs relationship database (part of this study focused on SPs and their assessments)**
   
   Six of the current well known references of software architecture patterns were selected and studied. The problems, conflicts, and limitations of those six resources regarding SP descriptions and documentation are presented in critique analytical fashion. Categorizations for all resources were conducted and compared. Relationships between all SPs within those resources and the ISO-9126 quality model have been studied and illustrated. Note that ISO-9126 has been replaced by ISO 25010 (the first part of which appeared in 2011 with a more complete version in 2014). It is an improved model of quality. Some of the Chapter 3 findings were slightly affected by this new standard. However, most of the findings are unaffected, as explained in Chapter 3.
   
   A database application has been developed to gather together all the investigation data to ease traceability of the above relationships, and to help developers quickly select best-fit SP for their problem context. Also, it has exposed (and also helps confirm) some of the issues regarding SPs in particular and SA in general.

2. **Opinions of the software developers concerning SPs**
   
   It is important to know what software development/project managers, architects, designers, programmers, etc. know and think about SPs. A survey was created and distributed to participants in six countries. The questionnaire targeted developers with over 5 years of experience. The findings and recommendations of the survey results were analysed and reported. One of the most significant findings of this survey is the strong agreement among
participants about the relationships between SPs and QAs database that has been developed within this research.

3. **Comparison between (theoretical academic research and real world development environment in industry and government), regarding SA and SAE**

To be able to judge, improve, and contribute to the arena of software engineering in general and software architecture in particular, I have conducted the research presented in Chapter 5 using two different methods (a survey and field study), both of which targeted software architecture descriptions, modelling, and evaluation of architectures. The gaps between the two worlds (the theoretical and the actual) are identified and discussed.

4. **Reference Architecture under dissection and proposing evaluation concept**

To demonstrate a proof or confirmation of the findings regarding the investigations, analyses, and work presented in Chapter 2 to Chapter 5, an analysis of a reference model called Real-time Control System (RCS) has been introduced to point out the critical and solid connectivity between SA, SPs, and QAs and how they affect each other. Also, an improvement and modification to an existing performance model, in order to generalise SA evaluation, is presented with some examples.

5. **New concepts proposed to be developed in future work**

The knowledge that I have gained from all the investigation, analysis, and effort surrounding the above four points merged to synthesise two important new concepts: System Architecture Evaluation (SysAE), and Software Architecture Quantitative Evaluation Framework (SAQEF). The main idea behind these two concepts is to provide more advanced evaluation techniques to take advantage of existing SAE methods, and to eliminate their drawbacks.
Chapter 2

Background

At each step, the process begins with a perception of the whole. At every step (whether it is conceiving, designing, making, maintaining or repairing) we start by looking at and thinking about the whole of that part of the world in which we are working. We look at this whole, absorb it, and try to feel its deep structure.

Alexander [2005, p4]
2.1 Introduction

The ever-increasing influx of data, evolution of technology, and the constant progression of systems from being software-intensive to ultra-large scale, strongly suggests there is a significant need for novel ways in which to construct, run, manage, and evaluate such systems. In software engineering, one of the main objectives, even as far back as the first software engineering conferences of the 1960s, has been the development of software systems which consistently satisfy a certain set of requirements, both functional and non-functional, Becker [2008]. The importance of software architecture under the umbrella discipline of software engineering has increased throughout the years. In fact, in the 1990s, this sub-discipline was a main focus of research, as predicted by Fielding [2000].

It is proper here to define system architecture before defining software architecture because the latter is part of the former. System architecture is comprised of three important components, which are software, hardware, and people. System Architecture as defined in the Series by the Software Engineering Institute (SEI) Group is: “a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of software architecture onto hardware architecture, and a concern for the human interaction with these components”, Bass et al. [2013].

What is Software Architecture (SA)?

Zachman [1996] provides an influential description of software architecture. “Architecture is that set of design artefacts, or descriptive representations (i.e. models), that are relevant for describing an object such that it can be produced to requirements (quality) as well as maintained over the period of its useful life (change”).

More recently, SA is defined as: “The set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both” Bass et al. [2013, p4].

The IEEE-42010 Principle extends the foregoing explanatory descriptions of architecture. According to IEEE-42010, architecture denotes: “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”.

Numerous other descriptions are available, each making a similar statement but using different expressions.

According to Bass et al. [2013], and from my experience as an air defence force officer in charge of a software development and testing team, there are three points of SA significance as follows:
2.1. INTRODUCTION

• Software architecture is critical not only in understanding a system’s characteristics, but also at the architectural level, where these qualities ought to be designed in and can therefore be evaluated.

• Software architecture lays the foundation with which quality can be achieved. This groundwork, however, will only be useful if the details (including implementation) are attended to just as closely.

• Software architecture can save time and costs significantly if it receives enough attention during the development and documentation processes.

As technology continues to evolve, software architecture as a practice faces new challenges. For example, making justifiable decisions is a feature pivotal to software architecture. To aid this process, a common solution is architecture evaluation.

What is Software Architecture Evaluation (SAE)?

“Software Architecture Evaluation has been proposed as a means to achieve quality attributes”, Babar et al. [2004]. So, the architecture evaluation in brief is, a form of artefact validation, just as software testing is a form of code validation, Clements et al. [2002b].

Hence, architecture evaluation can be defined as a process that judges whether an initial architecture description, or a series of candidate descriptions, represent the specified quality attributes, Qin et al. [2008].

Drawing these together, I propose a definition for SAE as: a procedure that consists of at least a specific method and an experienced team, in order to determine the level of quality requirements that are possessed by software architecture, or an architectural style, to verify if the architectural decisions are correct.

Software Patterns (now often referred to as styles) are important parts of architectural solutions; thus, we need to define SPs.

What is a Software Pattern (SP)?

“Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces, which occurs repeatedly in that context, and a certain software configuration, which allows these forces to resolve themselves”, Alexander [1979].
2.2 Initial research

Recent times have witnessed the emergence of software architecture evaluation as a significant practice and research area. These developments have prompted practitioners and researchers to create techniques, tools, and methods, which facilitate architecture appraisal, Overhage et al. [2007] and Breivold et al. [2012]. Further, system architecture affects related development processes and acts as a centre for development. Consequently, architects generally qualify as leaders of the majority of development groups. Indeed, architecture remains one of the leading professions within the technology realm, Schlosser [2017].

In addition, quality attributes are a prime consideration during software architecting, designing, implementation, and testing for any engineering product that includes significant software components. No attribute is totally independent; each quality attribute supports, hinders, or has no effect on others. Existence of the ‘no effect’ option has low probability within the software development domain. This is because of the nature of software components’ interactive processes. Also, the interwoven relationship between quality attributes makes the trade-offs process necessary to some extent. More so, it increases the importance and complexity of capturing required qualities on an architectural level. Therefore, if one misses important requirements in the architecture phase, one is also almost sure of missing the same requirements through the next phases of designing and implementation. As a result, desired quality in the end-product will be unrealised. It is therefore important to get the architecture and its description correct for a successful result.

On the other hand, self-adapting systems, such as Artificial Intelligence (AI), have their own specific/particular sets of requirements, such as the ability to adjust to operational context changes, Cheng et al. [2009]. With AI systems, in particular, symbolic computing has come to signify a host of concerns regarding their architecture. One point of interest is to integrate the use of non-numeric symbols into a computer’s main functions. In essence, the goal is to have machines visualise environments, reason about them, based on their knowledge, and develop the best plan to be executed. This type of system increases the complexity of the architecture and its evaluation, Meystel et al. [2002]. Thus, the reasoning of quality attributes as they relate to software architecture is an important aspect. Although, it should be noted that AI researchers are too focused on specific concerns, which do not, in particular, involve software architecture as it generally applies. There are, however, a few exceptions to this generalisation, one of which is the Albus reference model 4-D/RCS. The architectural framework of this reference model tackles a range of behaviours, awareness, and knowledge attributed to humans. This open/operational control intelligent system architecture has been utilised in a variety of applications: autonomous ground vehicles, cleaning and debarring workstations, control systems for postal service stations, and more, Albus et al. [1996]. A second exception, is Roy Fielding (AI researcher), who developed the Representational State Transfer (REST) architectural framework for the purpose of understanding and evaluating architecture through the architectural styles used.
2.2. INITIAL RESEARCH

in network-based application software, to satisfy the requirements of internet-scale distributed hyper-media systems, *Fielding* [2000]. Both architectures point to a strong relationship existing between SA and SPs.

In addition, every system needs to evaluate its software architecture to work properly. SAE methods are used to perform such assessments, which may also apply to Artificial Intelligence System (AIS).

However, it is general knowledge that software architecture is difficult to evaluate and compare objectively, *Fielding* [2000]. Thus, it is beneficial that requirements for quality attributes are identified in the early development phases of software, and are encapsulated within its architecture. The following reasons explain the importance of software architecture evaluation:

- The description of the software architecture is one of the first artefacts that can be evaluated and analysed.
- The problems that are discovered during SA development processes can be fixed at considerably lower cost than if they are discovered in the testing and/or deployment stages, *Clements et al.* [2002a]. In the former, all that is involved would be modifying notations in the architecture. However, in the latter, source code may need to be changed on a massive scale, inducing needless cost and delay. The utilisation of full automation and modelling techniques may help reduce these costs, if they are available.
- Architecture evaluation at an early stage helps open the communication paths between stakeholders and to develop a satisfactory system architecture which increases the success of projects.
- Architecture is at the centre of the development process. It includes decisions for the team structure, work division, configuration repository, documentation organisation, management strategies and development scheduling. An unsuitable architecture will cause a significant amount of disorder when it must be modified to address new concerns or defects that are uncovered in the early phase, *Qin et al.* [2008].
- The evaluation of software patterns (as a component of SA) aids developers in using and integrating them into other architectures, if need be, by utilising prior information on quality attributes.

A concern that emerges here, is the quality of the resultant architecture when patterns with differing quality attributes are combined. However, the relationships between software patterns and quality attributes have been analysed and identified in a scientific manner, *Freitas* [2009], *Kim et al.* [2006] who used Alloy-Analyser as a tool, and *Zayaraz* [2010].

Quality attributes, however, cannot be obtained in isolation, particularly within the context of complex systems. Whenever one attribute is achieved, another attribute is consequently affected. One such instance of this, is the relationship between performance and security. Typically, the more security requirements that are applied to a system the more time is needed to process security checks, which in turn means a decrease in system performance, *Bass et al.*
The current research derives from the significance of software architecture and its assessment, the worth of automation, the procedure for assessments, consistency of information and standardisation of notation, plus limitations of the current evaluation methods and techniques. Therefore, the outcomes are highly likely to add to the current body of software engineering knowledge.

In order to understand and improve Software Architecture Evaluation, there are some critical aspects that need to be addressed as reflected in the following questions:

- How is SA being described? What are the limitations of the current SA description methods?
- Do the current modelling methods have an effect on SA? What are the drawbacks and limitations of the current SA modelling techniques?
- How is SA being evaluated? What are the limitations of the current SAE methods?

The rest of this chapter addresses these questions.

According to Klassen et al. [1998], a Systematic Review is “a review in which there is a comprehensive search for relevant studies on a specific topic, and those identified are then appraised and synthesized according to a predetermined and explicit method”. The five common main steps in a systematic review are shown in Figure 2.1.

Figure 2.1: The main common steps for a Systematic review.

However, The Integrated DEFINition language 0 (IDEF0) diagram (Figure 2.2) and Table 2.1 draw the theme of the processes and strategies that been applied during this thesis research.

1The IDEF0 (in brief), is a functional modelling approach designed for manufacturing and logistics activities, in order to describe processes, functions, development, business, and analysis within the engineering field, Buede et al. [2016].
Table 2.1: This is the review strategy that has been applied in Chapter 2, followed by Chapters 3, 4 and 5).

<table>
<thead>
<tr>
<th>Steps of ‘evidence synthesis’</th>
<th>Comprehensive search and systematic review roadmap</th>
<th>Implantation and comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Preliminary research</td>
<td>In order to enhance SAE methodologies, the current SAE methods, gaps, and limitations must be identified. Thus, to scope the objective and to form specific questions, initial research should be performed.</td>
<td>Preliminary research has been performed, reported on, and discussed with, the supervisory panel and approved by them, through a Study Plan Report, on 30th January 2011 (12 pages) and (Thesis Proposal Review (TPR)-4th version,(44 pages). Some parts of the preliminary research were included in Chapter 2.</td>
</tr>
<tr>
<td>2 Objective</td>
<td>Is to combine strengths of critical comprehensive search processes and systematic review. It is to describe the state-of-the-art to answer questions regarding SA and SAE. It can address broad or narrow questions to produce ‘best evidence synthesis’ and can provide answers to those questions with critical descriptive analysis, either quantitatively or qualitatively.</td>
<td>A structured introduction and initial research are presented respectively in Sections 2.1 and 2.2, which will allow readers to assess quickly the relevance, quality, and generality of the review presented in Chapter 2.</td>
</tr>
</tbody>
</table>
| 3 Question formulation        | The rationale for the survey is based on specific questions. Questions can be open-framed or closed-framed. | • The research questions were formulated and justified based on initial research in Section 2.2.  
• SA and SAE are broad topics. However, several narrow subjects within (SA and SAE) were discussed, such as Formality, MDA, and Performance models, etc.  
• Three important open-framed questions were listed, each one has been answered through major sections, which are: 2.3 SA; 2.4 Modelling SA; 2.5 SAE. |
| Developing the research strategies and protocol | It’s an important step of the review process to identify the method and strategy on how the research will be conducted. Prior planning helps to increase consistency, integrity, and visibility. Four important phases need to be considered during the review strategies as follows:

1. **Critical**: assess selected information and articles.
2. **Analyse**: extract-opposing evidence from selected materials.
3. **Synthesise**, compare, and reveal the relationships between selected studies, concepts, or theories: differing characterisations and descriptions, etc.
4. **Evaluate and appraise** practical use of the selected approaches.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| I considered my Master research Almari [2010] as a pilot study for this review due to its relevance. The key points of the review strategy and protocol in this thesis are summarised in this table as follows:

1. The review title presented in Section 2.2. The reviewers include: my supervisory panel, and (external editor and proof-reader) as indicated in the acknowledgement section. The research duration is visible within the study plan report.
2. The objectives rationales pertaining to the questions and questions formulation are illustrated in Sections 2.1 and 2.2
3. Criteria for studies inclusion, search strategy, critical appraisal and data extraction, synthesise data, and reporting are all summarised in this table.

For versioning, bibliography management, and document retrieval, I have used several applications during this research such as Curio, Papers, Concept-Draw Office, Bibdisk, Texmaker, Office, Excel, Word). |
The goal here is to find as many primary and secondary studies that are related to the thesis question as possible. No limitation on research evidence, which can be included (e.g., primary and secondary research).

**The data selection concept in this thesis review follows a multistage iterative process.** The strategy used to quest relevant studies and sources include: journals (cover company journals), conference proceedings, scientific databases, grey literature (i.e., technical reports), books, the Internet, etc.

Sources of evidence have been searched using the following methods:

- Trial searchers using (search-string) various combinations of terms and keywords derived from the research questions using boolean ANDs and ORs.
- Consultations with my supervisor.
- Relevant studies references, strings, abstracts; and keywords.
- Relevant reliable websites.
- Relevant Conferences materials.

Also, randomized controlled trials that investigated the research questions are used.

"Current software engineering search engines are not designed to support systematic literature reviews. Unlike medical researchers, software engineering researchers need to perform resource-dependent searches, Klassen et al. [1998]. However, the case now is much more mature than the situation in 1998."
Sift the search result

The study selection criteria are proposed to classify those studies, which provide direct evidence and strong relations to the research questions.

Articles and information classification schema was based on the relevant subject (SA, SA modelling, and SAE). Applications such as Curio, Papers, Concept-Draw, Bibdisk, Texmaker were utilised during the sifting process in order to manage the inclusion process and to avoid duplications and confusion. The articles selection was based on the articles’ abstracts, key wording, and full text context that are relevant to the thesis questions.

**Inclusion Criteria:**

1. Any study that is relevant to the three main review questions regarding SA, SA modelling, and SAE.
2. Any study that addressed any of the research questions.
3. Any study addressed (SA and SAE) standardisation and automation processes in particular.
4. Empirical studies, experiments, surveys, models, standards, and frameworks that aimed to enhance and/or appraise (SA and/or SAE) methods.
5. Any information related to software patterns/styles documentation and evaluation.
6. Related information published in English language.

**The following types of materials are excluded:**

1. Unsourced information was excluded and removed.
2. Materials recommended as inappropriate by supervisory panel was excluded and removed.
3. Standards and frameworks that have been published after 2010 were excluded and will be recommended to be included in the future research.
4. Studies published in more than one journal/conference, the recent versions were used. Other versions have been excluded.
5. Studies published with the same aims, designs, methods, and results were excluded. Only the most common, reliable, popular, and complete studies were selected and included.
6. Unpublished articles and incomplete data were excluded.
| Page | Critical appraisal & data extraction | Data extraction: All the information needed that is satisfying inclusion criteria and relevant to the review questions was extracted. The data extracted include common information such as date of data, title, authors, publication details, additional notes available, etc.). Critical appraisal processes were carried out based upon the information relevant to the research questions, which include the context facet + articles facet, and contribution facet. Also, included are studies’ design, methods used, and tools were. A Critical appraisal checklist with main five questions was created and employed. Each main question of the checklists comprises sub-questions:
1. Does the review address clear research questions?
2. Do the reviews’ results help to achieve the thesis main objectives?
3. What are the results of the review?
4. Are the review results providing a valid answer to the research questions?
5. Will the research results benefit the engineering field?

Supervisors reviewed all the research information, processes, and methods. They pointed out to any incorrect actions or issues within any aspects of the research. Regular meeting were scheduled and executed during the research. Re-evaluating the selected studies after screening was conducted on iteration, in order to check their consistency to inclusion criteria.

<p>| Synthesis narratively (and/or) statistically | Trends in the literature, knowledge gaps, and clusters identified. Qualitative or quantitative synthesis of study results, where possible, using appropriate methodology or analysis. | Describing the findings in critical fashion with a positive argument towards the research area and question was conducted. The important aspects, issues, gaps, comparisons, method mapping, and limitations around the three main subjects which are SA, SA-modelling, and SAE, were logically and critically discussed and identified, as illustrated in Chapter 2. Visualising the findings through figures, tables, and mapping mechanisms were presented. |</p>
<table>
<thead>
<tr>
<th>9</th>
<th>Report</th>
</tr>
</thead>
</table>

The report should describe and categorise available data relating to the research topic and questions, identifying knowledge, advantages, disadvantages, and gaps; identifying implications on the thesis research schema.

Mostly narrative and qualitative synthesis study results (e.g. comparative and critique analysis) were used, in order to answer the review questions. Implications for this thesis were illustrated and summarised in Sections 2.6 and 2.7. The knowledge gaps and limitations were presented in Chapter 2, and recommendations for future research are made and reported in Chapter 7. Reporting the review in Chapter 2 was designed and catalogued carefully in order to point out to the current research problems and to be able to answer the research questions, while keeping the aim and structure of the whole thesis concrete, which is justified in Chapter 1 and Sections 2.1 and 2.2. This thesis contribution to the identified problem is presented in Part II.
2.3 Software Architecture Description

The study of software architecture is a complex field and involves the study of design principles and design patterns used in the creation of a software system. The design and implementation details of software architectures can be represented using different methods that can be classified as; formal, semi-formal, and informal methods. Each of these methods has some advantages and certain drawbacks. However, the basic principle behind these methods remain the same which is to provide a way to describe the structural elements of a system along with any supporting information needed. In general, the design representations of the architecture are also known as high level design documents. *The aim of this section 2.3, is to explore the way architecture described within different methods and to compare between them within the contexts of SA and SAE. I conclude the section with a study of common formal methods and their relations with SA and SAE.*

2.3.1 Brief Analysis of SA description methods

The architecture of any software application is an integral part of its design and deployment. Therefore, it is extremely important for any application to have well-defined descriptions outlining the architecture. Thus, the study and description of software architectures is necessary aspect of the software life cycle.

---

2 In the context of this thesis terms such as architecture and design are including styles and patterns.
There are several different ways in which software architecture can be viewed depending upon the purpose and target audience.

There are three main methods for describing SA, as illustrated in Figure 2.3.

Figure 2.3: The three main methods for describing SA.

1. **Informal methods** uses languages/notations such as textual, English language, and general purpose diagramming.
2. **Semi-formal methods** uses languages/notations such as UML, and SysML.
3. **Formal methods** uses languages/notations such as ADLs, and Z.

These methods are used to describe various kinds of software architectures in terms of its structure, behaviour, components, and the relationship between them. This section explores the ways in which each method of description differs from others. Also, it is to focus on the common formal methods for describing software architectures.

There are several problems that need to be addressed during the development of a software application. If the size of the application is small, the computational problems, which are encountered can be solved by using regular algorithms and data structures. However, larger applications are rarely so simple or one-dimensional and are often made up of large interconnected structures, which need to work together as a system. For example, most of Command, Control, and Communication (C³) systems include mission planning and direction subsystem, mission execution subsystem, and surveillance subsystem, which are connected to each other through various interfaces and, hence, support each other, *Hughes Company* [1993].

The same applies on Command, Control, Communications, Computers, and Intelli-
Also, the same concept exists on banking software applications, which are made up of core banking modules that work together with an internet banking modules and credit card handling systems.

These complexities are inevitable to ensure that the application delivers the necessary functionality flawlessly, remains reliable, reusable and is easy to maintain. However, the architecting, designing, and testing of such complex systems poses several challenges. As the size of the system increases, the development related problems increase manifold and with the increasing size and complexities of present day software systems, design problems can no longer be sidelined, Safwat et al. [2015].

The emerging field of architecture description concerning different aspects, such as defining the high level system components and the connectivity between them, predicting and formalising the expected behaviour of the system and defining the overall abstract structure of the system that identifying the interconnected components. These processes contribute towards solving problems like translating a system from requirements to design, creating architectural descriptions and ensuring that the architecture will stand up to the requirements, Dick et al. [2017].

In the context of this thesis the Diagrams and Models differed from each other. Whereas, Diagram is a form of graphical representation that explains the architecture using any format and notations alongside a natural language, without following any standards, languages, or specific semantics. While Model is an abstract that follows a standard notation to describe some aspects of the architecture, such as its components structure, and how they interact with each other.

• The informal methods for describing software architectures (e.g. English, general purpose diagramming) mostly used to explain the architecture in terms of simple boxes and lines where boxes represent the components and lines represent the connections between them, with some textual comments that does not follow any rules or specific notations.

The informal description of SA is easy to develop, understand, and interpret. However, some of the common limitations arising due to the use of informal languages are mentioned below:

1. Ambiguity: Informal diagrams, notations, and the use of natural languages result in a lot of ambiguity regarding various factors, such as the meaning of connectors, their directionality and related associations. In addition to, the way data and control information flows, despite the components and connectors being represented as separate presentation and visual forms. Often, these ambiguities can result in confusion and non-uniform interpretations, which can lead to misunderstandings.
2. Communication gaps: Software development is no longer an individual job but often
requires large teams interacting with other each to produce separate pieces of artefacts, which together form the system. In such scenarios, smooth flow of information and good communication is very important and the ambiguity of informal methods hinders both.

3. System validity issues: From the development perspective, it is important that the architecture is validated early, to ensure that it meets the specifications and fulfils the requirements. This ensures that the architecture is translated into a design, and then can successfully be translated into code. It is very difficult to properly validate an architecture which is described using informal methods. Even assuming that the information is complete and accurate, it is still difficult to arrive at a precise validation model due to the lack of scientific or mathematical notation to measure the completeness and quality of such systems, Pressman [2006] and Qin et al. [2008]. This might lead to a system failure at a later point of time. This becomes especially problematic if the project development is being executed from multiple geographic locations.

4. Inaccurate behaviour description of architecture: The behaviour description of architecture deals with the functionalities of different units, communication between them and their validity. Hence, it is a very important part of the overall description, but informal methods fail in several aspects to describe SA behaviour adequately, due to various factors like ambiguity, translation of diagrams into analytical models, tool based support and automation process. All this can lead to communication gaps, system deadlocks, and invalid systems.

5. One of the main differences between informal methods and other methods, is that the system and architecture verification is done manually with informal methods. Whereas in the case of other methods, verifications are (or could be) automated and standardised even though the effort involved would be greater, Rushby [1993a]. Taking into account that, a manual ‘verification’ may involve incorrect interpretations of symbols within diagrams, because sometimes a box might be interpreted as a component and lines might be assumed to represent flow/order, while the architect meant boxes to merely indicate concepts and arrows to indicate data-flow. Thus, the communications between stakeholders is an important factor to have common interpretations.

Therefore, the informal approaches do not provide a good foundation for describing and evaluating SA, even though they can be useful in the initial stages of a project life cycle.

• **Semi-formal methods** rely heavily on standardised notations/languages (e.g. UML and SysML) that prescribe the architecture and follow rules to apply them, Bass et al. [2013]. Recently, the Semi-formal languages have become more advanced and supported by tools. Their level of formality is increased and they can be automated to generate code such as using XUML language with a model compiler, or the Artisan tool, that generates different

---

3 Artisan Studio is an integrated suite of modelling tools previously developed by Atego Inc. which is now part
2.3. SOFTWARE ARCHITECTURE DESCRIPTION

types of code (e.g. Java, C++ and Ada) from the UML/SysML models.
The high level models created by using these languages can be of one or more different kinds, which are used for visualising the architectural designs of the system. For example, Use Case models, Business Process models, Object models, State models, Deployment models, and Component models. Each of these models describes one or more aspects or points of view of the system architecture through diagrams, often intended for particular group(s) of stakeholders. For example, the Use Case model can be represented by Use Case diagrams, which is meant for both the user and developer. Similarly, the Deployment diagram can be used to represent the Deployment model, which is useful from the system engineer’s perspective etc.

Semi-formal models are more generic and can cover different views of the architecture, even though they could be limited during the analysis phase. Also, they can be annotated by a natural language and automated at the same time.

However, two points need to be considered during the utilisation of semi-formal models as follows:

1. The generation of code is mostly incomplete because accurate and precise models are needed. Hence, the generated code may require some amendments after the generation process, due to different reasons, such as models accuracy, tools maturity, and the size of the system.

2. Semi-formal methods use natural language features, which could be good or bad, depending upon the architects and the accurate utilisation of the natural language in the correct manner and places, when constructing architecture descriptions.

In addition, there are some comments in the literature regarding semi-formal notations/languages in general and the UML in particular such as, it is not completely sufficient to eliminate the presence of ambiguity as models can contain only a limited amount of information and expressions. Considering the size of specification documents for a complex system, it is almost impossible to represent them with complete accuracy, Pressman [2006] and Qin et al. [2008].

Furthermore, Semi-formal methods lack precise descriptions for SA behaviour. Even though the semi-formal diagrams, such as (activity and sequence) diagrams are useful in predicting the behaviour of systems, but “cannot represent time constraints effectively”, Ribeiro et al. [2016]. Also, they are insufficient to describe computational data and to simulate real-time dynamic behaviour. All this can lead to communication gaps, system deadlocks and invalid systems, Rozier [2016].

I argue that the above comments are partially correct with respect to UML, where “UML modelling tools provide poor support for composite state machine code generation. Generated code is typically complex and large, especially for composite state machines. Existing
approaches either do not handle this case at all or handle it by flattening the composite state machine into a simple one with a combinatorial explosion of states, and excessive generated code”, Badreddin et al. [2014]. I agree that the size of the system could be a challenge for Semi-formal approaches in regard to accuracy, features and precise code generation, which been noticed during my inspection for the speed controller system artefacts with Artisan tool.

Generally, the code generation process for UML models, are dependent on the properties of a specific tool generator. In order to make generation process independent and compatible with several tools, the code generators need a “uniform input model”, which a noteworthy contribution towards standardisation concept and UML models transformation introduced by Noyer et al. [2014].

Recently, semi-formal languages have evolved dramatically and they can model the whole system including its behaviour and generate code from those models with more precision. Also, they have been involved in SA frameworks (e.g., DoDAF), formal methods (e.g., AADL), requirements, automation analysis, code generation, management processes. There are hundreds of semi-formal development in different domains, such as (but not limited to) the work done by Lamancha et al. [2010], Silva Melo et al. [2014], Pereira et al. [2015], Hilken et al. [2015], Lugou et al. [2016], Ribeiro et al. [2016], and Ribeiro et al. [2017]. Indeed, Semi-formal methods are not as precise as Formal methods. However, Semi-formal approaches do have the strength over the Informal languages, such as, understandability. So, they are understandable to human, and their models can demonstrate different architectural views. Also, they do have some of the Formal methods strength regarding machine readability.

Furthermore, They do not have the same ambiguity as Informal methods. Also, it’s not hard to learn, develop, and understand. As a result, it is a good foundation for describing and evaluating Software Architecture (SA).

- **Formal methods** are used to express architectures formally. One way is to describe SA in terms of Component and Connector (C&C) and configurations that carry computational information and provide the foundation for development of analytical functions. Such languages are (but is not limited to), the AADL, ACME, Arch-Java, Koala, and MontiArc Automaton Architecture description language (MAAADL), Wortmann” [2016].

If we take as an example the AADL and the MAAADL as formal languages, they are considered to be an efficient to analyse and create architecture descriptions based on formal notations and tools, Medvidovic et al. [2000] and Wortmann ” [2016]. The AADL is a modelling language to model both (software and hardware) components, and associated properties, whereas MAAADL focuses on modelling SAs logically.

According to Wortmann” [2016], “the MAAADL modelling infrastructure comprising the concepts, ADL, state-based behaviour language, model transformation and code generation
2.3. SOFTWARE ARCHITECTURE DESCRIPTION

framework to enable multi-platform generative software engineering”. More analysis of formal methods is presented in the next section.

However, if we compare semi-formal and formal, we find that each one has different advantages. Formal languages are powerful during syntax and semantics analysis. Some of the main advantages of formal methods in describing behaviour are mentioned below, Qin et al. [2008]:

1. Formal methods are more concise and accurate than other methods, which able inventors or designers to express their specific concept through its semantics, and hence support new designing approaches to be manifested.
2. Formal methods are more effective in describing aspects such as behaviours and patterns, thus they are cognitive to behaviour analysis and creation of rules.
3. Due to the absence of ambiguity, it is possible to validate the system architecture beforehand and also measure it for quality parameters.

In general the above points is correct, but it is vary from language to another. For example, the formal semantics and executability are two limitations of AADL’s as examined by Ölveczky et al. [2010]. However the AADL were evolved during the last few years and its semantics becoming more mature, Oquendo [2016].

Challenges hindering the utilisation of formal methods are:

1. hard to learn;
2. hard to develop;
3. hard to understand by all stakeholders;
4. need an expert, who may not always be available;
5. high cost;
6. time consuming.

Considering the above factors along with, the fact that Semi-formal and Formal methods description play an important role in the application of software architecture description and modelling approaches, they are being hailed as the way forward.

More discussion about Formal methods is represented in section 2.3.3.

See the three notation representations in Table 2.2 that are describing a section of the Pipe-Filter pattern, in order to better visualise the argument above.

2.3.2 Views of SA description

The basic idea behind the architecture of a system is that its elements can either be broken down into smaller parts or combined to form bigger parts. Most of the development in the software field is currently carried out on the basis of Object Oriented Development (OOD) and the architecture is also described according to these concepts. This field of study is known as
Table 2.2: Formal, Semi-formal, and Informal methods for representing a section of Pipe-Filter family.

<table>
<thead>
<tr>
<th>Informal description (POSA-V1) Buschmann et al. 1996</th>
<th>Formal description with Z language Shaw et al. 1996</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class: Filter</td>
<td>filter_id: FELTER</td>
</tr>
<tr>
<td>Responsibility:</td>
<td>In_ports, out_ports: P PORT</td>
</tr>
<tr>
<td>• Gets input data</td>
<td>alphabets: PORT → P DATA</td>
</tr>
<tr>
<td>• Performs a function on its input data</td>
<td>states: P FSTATE</td>
</tr>
<tr>
<td>• Supplies output data</td>
<td>starts: FSTATES</td>
</tr>
<tr>
<td></td>
<td>transitions: (FSTATES × Partial_Port_state) ↦ (FSTATES …)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Semi-formal description with UML graphical Representation Clements 2003.</th>
<th>—Filter_State—</th>
</tr>
</thead>
<tbody>
<tr>
<td>f: Filter</td>
<td>f: Filter</td>
</tr>
<tr>
<td>internal_state: FSTATE</td>
<td>internal_state: FSTATE</td>
</tr>
<tr>
<td>pstate: Port_state</td>
<td>pstate: Port_state</td>
</tr>
<tr>
<td>internal_state ∈ f.states ∧ dom pstate = ...</td>
<td>internal_state ∈ f.states ∧ dom pstate = ...</td>
</tr>
<tr>
<td>∀ p: dom pstate -ran (pstate(p))…</td>
<td>∀ p: dom pstate -ran (pstate(p))…</td>
</tr>
</tbody>
</table>

Object Oriented Analysis Development, which is most commonly represented by specification languages like the Unified Modelling Language or UML, Booch [2005].

There are different kinds of views, each of which can be made up of one or more models. The most popular among these is the 4 + 1 architecture views, which is made up of logical, development, process, and physical views. Each represented by a separate diagram, and using a different kind of visualization for its architecture to report it to different stakeholder communities, Obbink et al. [2002].

In most cases, the description of any software system can be summarised using three important models: functional model; object model; and, the dynamic model. Figure 2.4 illustrates this, showing that the 3 models together provide sufficient information about the whole system. Also, each of these views can be divided into several sub-views that can form the whole parent view within the same context. Each of these models is described below:

1. Functional Model: This is primarily meant for the end user and describes the system’s functionality by separate use cases through Use Case diagrams. The overall structure is described in terms of functions and processes.
2. Object Model: This model represents the system in terms of objects, relationships between them, properties of the objects, and the functionalities provided by overall objects. They (objects) are modelled after real-world entities and this model primarily depicts the information through class diagrams.
3. Dynamic Model: The focus of this model is on the functionalities provided by the system and its behaviour during normal operations and state transitions. This model can be
represented by activity diagrams, state machine diagrams, sequence diagrams, etc.

![Functional Model (e.g., use cases)](Diagram)

![Dynamic Model (e.g., state machines)](Diagram)

![Object Model (e.g., class diagrams)](Diagram)

System Description

Figure 2.4: General prospective of the model types for describing a system.

Some of the common terms used in the modelling world are actors, scenarios, and use cases. Here, an **actor** is any participating entity of the system, such as a person or a computer. Whereas, a **scenario** is a specific sequence of actions which yield some result, and the **use cases** are a set of functional scenarios which can either be successful or fail and are performed by an actor to achieve the goals Larman [2004]. The relationships in the diagram can be termed as associations, dependencies, generalizations, etc.

### 2.3.3 Formal methods and languages

Formal methods are very useful in dealing with the shortcomings of the informal methods such as ambiguity and inconsistency. According to Kelly [1997] and Storey [1996], they generally consist of logical and mathematical methods to create specifications and design of a software system and also to construct and analyse these systems, Schumann [2001]. Also, formal languages are constructed to be to be completely accurate and precise regarding every aspect of the system. The formal languages are developed on the basis of formal techniques and mathematical theories and formulae. The formal methods can be applied at any point in the software development cycle to describe the system at whatevergranular level is desired.

Formal methods can be applied starting at the highest (most abstract) level to the bottom, to ensure that all the critical properties and aspects of the system are described accurately and can be analysed with precision. Due to the application of mathematical theories in formal languages, the system is consistently measurable. Applying formal methods at various phases in the process ensures smoother transitions. For example, the design can be created based on mathematical principles and the development can be done according to these accurate designs using standardised principles. This process ensures that the overall system is reliable and the communication gaps are eliminated between developers.

All the advantages mentioned above and in Section 2.3.1, are to ensure that the system can be verified against the requirements before the development or implementation cycles commence. Also, due to exact definition of system boundaries and constraints, it is also possible to
trace the verifications backwards, i.e., to check if code conforms to design, design conforms to requirements, etc. There are several tools available to check if the constraints and boundaries are consistent. For example the ACME-Studio tools can be used to check for inconsistencies and anomalies in the architectural model. It is also possible to automate the validation checks for such systems, Qin et al. [2008]. Such automation techniques can be based either on: a) Automated Theory Proving, wherein the automation system takes the description, axioms and rules as inputs and generates a logical validity proof as output; or b) Model Checking wherein the system’s validity is verified by going through all the possible stages that the system can go into and check whether the system’s properties are valid at each of these stages. The concept of Model Checking has traditionally been used in the hardware field extensively and has recently started spreading into other fields, Klarlund et al. [1998].

Considering all the above factors, usage of formal methods is very important especially in systems which require high levels of quality, such as security and confidentiality, to ensure bug-free software. By using formal methods from the initial requirement and design stages, the errors can be reduced thereby saving development and implementation rework efforts. It is not necessary to use formal methods at every stage of the project and in most cases partial formalisation is sufficient. Both Formal and Informal methods have their own shortfalls. However, studies have indicated that using formal methods is the best way to prevent bugs and produce high quality softwares, Rozier [2016], Qin et al. [2008]. With proper usage of formal methods can yield applications of high integrity and accuracy, thus the cost can be high initially. Also, formal methods can be used not only in development projects but also in various other types of projects like enhancement projects, testing projects and maintenance projects. There have been tremendous improvements in the area of formal method usage. Figures 2.5 and 2.6, demonstrate how formal methods have evolved since 1970’s until 2010, Schumann [2001] and Foster [2011].

![Figure 2.5: Evolution of formal methods.](image-url)
2.3. SOFTWARE ARCHITECTURE DESCRIPTION

2.3.3.1 Common formal methods and languages

A few common formal methods and languages are outlined below:

1. Petri Net: This was invented by Carl Adam Petri in 1962, which is used to describe distributed systems in a mathematical way.
3. Actor Model Method, Hewitt et al. [1973].
5. Z Notation: This theory proposed by Jean-Raymond Abrial in 1977, Leavens et al. [2006], which is a computational language based on set theory and calculus and is used for describing systems formally.
7. Architecture Description Language (ADL), Dissaux et al. [2005].
8. Domain Specific Language (DSL), Fowler [2010].

All of these methods are efficient in dealing with many of the problems associated with informal architectural descriptions and also provide extensive technical capabilities. However, they have their own limitations that prevent them from being used extensively.

Some of the main problems associated with formal methods and languages in general are:

1. An essential observation is that current ADLs, such as AADL, ACME, xADL offer support for various types of architectural elements, but do not support the architectural views in a way proposed by ISO-42010, which minimise the standardised aspect, BouckÁ et al. [2008]
2. Since most of the coding while developing a software application is done manually, the gap between requirements, specifications, and the code is still wide. However, new approaches such as, Model Driven Architecture (MDA) and Model Driven Development (MDD) that focus on automation can help in lessening this gap.
3. The integration process of software components during its development using formal methods is difficult and time-consuming, due to the rigid mannerisms of the formal methods, which is evident with the difficulties involved in learning and implementing such methods (e.g., Z notation). However, formal methods advantages can be gained in the long-term, Gogolla [2004], Qin et al. [2008], and Zamansky et al. [2015].

2.3.3.2 Applicability of formal methods in the software Life Cycle

A software development cycle is not a single process but consists of several phases beginning from requirement gathering to the maintenance of the finished product. These phases together contribute towards the development cycle. There are several approaches and methodologies available to conduct a lifecycle, the discussion of which is beyond the scope of this research. Formal methods can be used during any of the phases of development to achieve whatever degree of formal description is needed. However, it is important to use tools which support automation as it is one of the core requirements for the best usage of formal methods, Xia et al. [2012]. The following section illustrates the degrees of formality and scope during software development.

2.3.3.3 Degree and scope of Formal methods

The four levels of formalisation degrees mentioned by Rushby [1993] are described in Table 2.3.

<table>
<thead>
<tr>
<th>Level</th>
<th>Use of Formal Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This level involves writing of specifications with the help of natural languages, diagrams and pseudo-code. The entire process is informal and no formal methods are used at any point.</td>
</tr>
<tr>
<td>1</td>
<td>In this level, certain portions of the requirement and specifications are supplemented with the help of mathematical theories even though analyses and proofs are still carried out in an informal manner. This level reduces the ambiguities and is more accurate as compared to level 0 while still maintaining a level of flexibility. This also ensures compactness and allows for a smooth integration of design and development.</td>
</tr>
<tr>
<td>2</td>
<td>This level introduces the formal specification languages along with a certain level of support from tools, which provide various functionalities such as syntax checking, type checking, interpretations, animations etc. Proofs are still informal although they are enforced strictly. Also, some methods (e.g. Taylor 1990 and VDM/SL Jones 1990) allow performing of formal proof-checking manually in order to obtain the best of both worlds.</td>
</tr>
<tr>
<td>3</td>
<td>This level makes use of complete formalisation with formal specification languages being employed at various levels with the help of supporting infrastructures. It also formalises proof-checking using mathematical theorems. Techniques such as Theorem Proving, model checking and proof-checking are used in this level. Some common examples of this level are the work done by Bayer et al. 1988, computational logic for Applicative Common Lisp (ACL2) Kaufmann 1998 etc., where ACL2 represents a first-order programming language based on Common Lisp.</td>
</tr>
</tbody>
</table>

For all the levels mentioned in Table 2.3, the formal methods utilization is dependent upon the project goal and requirements, which determine if any of the above levels need to be used for
2.3. SOFTWARE ARCHITECTURE DESCRIPTION

a project or not. In most existing cases, level 0 or 1 are sufficient to describe the architecture, but are insufficient to automate the process and evaluation analysis. Levels 2 and 3 are considered to be sufficient for code automation and also for evaluation analysis, bearing in mind that there are several problems involved in applying these techniques for the first few times, such as the complexity of their implementation. Some of the common scenarios where formal methods can be used are:

1. Initial project stages: Usage of formal methods in the initial stages are hugely beneficial as they reduce the possibility of mistakes early on, thus save valuable time and rework effort. Studies have shown that fixing a defect post-development costs several times more than the cost of the effort spent in avoiding it, Pressman [1997] and Bendik et al. [2016]. The formal methods can also be used for verification of flows between different stages. They can be used in the early stages to create a traceability matrix between the client requirements and actual designs. In order, to ensure that the developed code is in-sync with the requirements of the clients. Though, these matrices are often not very beneficial from the client’s point of view, but they are extremely useful from the developer’s perspective.

2. Verification of system components: Rather than verifying the entire system, the important components of the system can be identified, then verified as stand-alone modules using these methods even though it is important not to overlook other less important modules as well. Typical candidates for such core validations are communication components, security components and audit components.

3. Verification of system functionalities: Again, the important functionalities of the system can be identified and verified rather than the system behaviour as a whole. Some typical examples include safety features and termination features. Also, in most cases, it is sufficient to prove the absence of failures such as deadlocks and security lapses rather than proving the success of each and every functionality, Schumann [2001].

2.3.3.4 ADLs as example of formal description

ADLs, are considered to be an efficient way to analyse and create architecture descriptions based on formal notations and tools, Medvidovic et al. [2000]. An ADL in its simplest form *it is a computer language used for describing architectures*, even though there is a lot of ambiguity about various aspects of an ADL, such as its purpose, scope, function, and interchangeability. Hence no exact approved definition of ADL is available, Clements [2003]. However, according to the IEEE-42010:2011, the architectural language defined as “any form of expression for use in architecture descriptions”. Hence, there has been a general agreement on certain aspects of architecture description and this has led to the development of a common, second generation ADL known as ACME which serves as a common platform for architecture description and which allows several related architecture analysis tools to reside under one roof.

*After ACME language appeared, some advances in SA description had improved such as:*

- The new ADL for describing SA of a System-of-Systems (SoS) named SosADL, which
CHAPTER 2. BACKGROUND

is supporting automated analysis and it’s associated with its toolset, Oquendo [2016].

- Architecture Analysis & Design Language (AADL) which considered as a complete language for designing both the software and the hardware and it’s supported by its toolset. AADL architecture models can be used for code generation, documentation, and analyses, Feiler et al. [n.d.].

As we can see from Table 2.4 and Figure 2.7 that AADL is more advanced than ACME. Actually AADL language incorporates all ACME components in its specifications and more. For example, A device in AADL could have logical connections via ports to a software components, as well as physical links to a processor through a bus, which is impossible in ACME language.

Table 2.4: General comparisons between ACME and AADL main components.

<table>
<thead>
<tr>
<th>The seven basic elements of ACME known as core constructs, after Garlan et al. 1997:</th>
<th>AADL main elements, after Feiler et al. n.d.; Feiler et al. 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Components: represent the individual elements of a system as described previously.</td>
<td>1. Component Type: defines a component’s external attributes and its interface elements.</td>
</tr>
<tr>
<td>2. Connectors: represent the relationships and connections between the various components of the system as described previously.</td>
<td>2. Component implementation: defines a component’s internal structure.</td>
</tr>
<tr>
<td>3. Ports: represent the interfaces of the individual components to the external world.</td>
<td>3. Package: supports categorising AADL elements into labelled groups.</td>
</tr>
<tr>
<td>4. Roles: represent the connector interfaces to the external world.</td>
<td>4. Property Set.</td>
</tr>
<tr>
<td>5. System: this is a combination of components and connectors. This also contains a description of the overall structure along with information about the hierarchy.</td>
<td>5. Annex Library.</td>
</tr>
<tr>
<td>6. Properties: represent fields to store system information, and also the information about various elements of the system, such as components and ports.</td>
<td>Elements 4 and 5 allow a developer to extend the language based on specific requirements.</td>
</tr>
<tr>
<td>7. Representations: describe the lower-level details of an architecture element by means of sub-architectures. Any element that needs to be described in minute details, is described with the help of a separate architecture description, which in turn acts as a sub-system of the parent.</td>
<td></td>
</tr>
</tbody>
</table>

There are several ADLs available for architecture modelling purposes in the field of engineering. Some are general purpose modelling languages and some are domain specific where the language and tools are customised to work with specific domains only. In either case, the focus is on the high level details as opposed to minute deployment specifics, so that the abstract designs and descriptions can be created according to requirements.

More Analysis of ADLs family and their design goals can be found in Appendix B, Section B.3.
2.3. SOFTWARE ARCHITECTURE DESCRIPTION

2.3.4 ACME in brief

This section introduced here as an example, to illustrate the advantages and disadvantages of one of the ADLs. In order to support our further discussions within the next chapters.

The main aim of ACME proposed by Garlan et al. [1997] is to provide a flexible and interchangeable environment for architecture description. It allows the integration of several ADL tools with different purposes under one roof in order to provide a platform for interchangeability. Along with interchangeability, ACME also aims to fulfil the following goals:

1. To provide a platform for the development and visualisation of new architectural patterns, using tools like ACME-Studio, which will also support architecture analysis.
2. To allow development of more domain specific ADLs.
3. To standardise ADL conventions and notations and to serve as a platform for standardisation.
4. To provide human readable descriptive expressions Shaw et al. [1996], and Mavridou et al. [2016].

ACME provides support for four architectural aspects:

1. Structures: to support the composition of a structure into its constituent parts or elements.
2. Properties of interest: to provide all the information about a system and its elements which is needed to visualise the high level architecture of the system and to understand its behaviour in terms of functional and non-functional aspects.
3. Constraints: to provide information about the boundary conditions and also acts as an
indicator of the way the system can evolve over time.

4. Types and styles: to provide types and styles definitions for the architecture descriptions.

### 2.3.4.1 ACME design trade-offs

As is the case of initial development for any language, conflicts between goals during the development could exist. This is applicable to ACME too. For example, the first conflict is in the design goals, where goals (one and four) mentioned in the previous section clashed with each other. This conflict arose due to the fact that the aim of ACME was to provide a language that is human-readable and easy to understand. At the same time, it should be in a machine readable format. Taking into account that human usability is an important factor, making a language user friendly can hamper machine automation capabilities. This is still far off in regard to the current ACME specifications and technologies.

The second conflict is the need to keep the language as simple as possible while making it rich and expressive to provide more flexibility. The simplicity and elegance was required in order to ease the visualisation of an architecture for the intended audience. While the rich and expressive features were aimed to support automation and standardisations, ACME has compromised by taking a middle path wherein the expressions are rich and expressive only when needed, otherwise keeping it as simple as possible, which is not a proper schema for full automation or deep analysis.

Addition to the seven basic elements of ACME mentioned in Table 2.4, ACME also has some language constructs that are detailed below:

1. Design element types: describe the structure and properties of design vocabulary items.
2. Property types: provide more information about various properties and their types.
3. Design invariants: describe boundaries which must be followed during design creation.
4. Design heuristics: provide ways for effective design creations.
5. Design analyses: computed functional values specified either in the ACME predicate language or as external functions.
6. Architectural styles: made up by other constructs mentioned previously and summarise all the aspects of the design required to create an appropriate architecture.

ACME is one of the good candidate languages for describing SA, which can serve as an ADL by itself by providing a construct that contains information about the system architecture along with information about its components, connectors, types, substructures, properties etc. It can be treated as a representative of the ADL family which describe the architecture of a system in terms of components and connectors. ACME’s architectural style, like the Pipe-Filter style, its Roles and Ports together with the component relations are described in detail within ACME and built-in within the ACME-tool. The properties in ACME are usually name-value pairs and the entire structure is represented in textual format, which has many advantages over the common C&C diagrams created by informal methods. However, in general ACME is easy to use and be
understood. It also can provide a lot of expressions and customised solutions in a constrained manner, even though it's lacking some of the flexibility and simplicity of general purpose ADLs. However, the use of ACME is generally accompanied by the use of some supporting tools which provide automation and graphical facilities. Another example (but not limited to) of a good candidate language for describing SA is the AADL, which provides support for error analysis, and its semantics possess well suited mechanisms that provide more reliable code generation among other key features.

2.3.5 Conclusion

Section 2.3 explored the differences between formal, semi-formal, and informal methods for describing architecture. Based on the studies conducted so far, formal and semi-formal methods are the descriptions that seem to be the best in order to describe and analyse architecture of software systems, including styles and patterns. Also, it shows that semi-formal methods have more advantages and less limitations than formal methods from most of all stakeholders viewpoints. not just from developers viewpoint.

2.4 Model driven approaches and Architecture

The description of software architecture, notations, and styles is closely related to the concept of model driven approaches, which all are an integral part of the software development process. In this Section 2.4, modelling methods are explained.

There are several approaches towards SA development that utilise a model driven concept. Model Driven Development (MDD)\(^4\), is an approach that deals with software complexity by making software models primary artefacts of the software development process. While Modelling Development Engineering (MDE) is an approach to software development that uses models as primary artefacts, from which code, documentation and tests are derived, Rech et al. [2009].

All the model driven development methods have certain similarities and certain unique features. A summary of the main differences between modeling approaches is visualised in Figure 2.8.

The basic idea behind the approach is to segregate the application specifications from the underlying technical details. To explore each of these model driven approaches in details is beyond the scope of this thesis. Hence, this section aims to explore and analyse the different flavours of model driven approaches and the role of patterns and quality attributes with respect to these model driven development methods, in the context of SA.

\(^4\)MDD includes all software modelling techniques.
2.4.1 Model driven software development

Over the years, several different approaches have been adopted by the software community towards the design and development of software systems. Although each approach has its advantages and disadvantages, concepts like abstraction and object orientation have always remained in vogue. Recently, the focus has been on a new approach towards software development which utilises models to create and support software development. This is known as Model Driven Architecture (MDA), which manifests itself in several forms. The main advantage of MDA is the high level of abstraction and platform independence achieved by the use of models, thereby eliminating any tight coupling with specific programming languages.

The general common manifestation of this approach is Model Driven Software Development (MDSD), which is implemented in different ways in several domains. *MDSD is considered to be a more accurate way of describing MDD and its use has increased dramatically in the last few years, Stahl et al. [2006], Pons et al. [2012]*. All the model driven approaches are bound to concepts like Object Oriented Methods and UML. In almost every kind of model driven approach, initially a domain model is created which is then translated into a meta-model, then converted into code generally by using some kind of generator, *Rech et al. [2009]*. The generation of models can be done with the help of a language like Executable UML (\(\text{\textcopyright}^\text{TM}\) UML), which
2.4. MODEL DRIVEN APPROACHES AND ARCHITECTURE

focuses on semantics and helps people utilise existing knowledge repositories to create precise systems. To identify the ways in which the various modelling approaches vary from each other is beyond the scope of this thesis; however, several valuable research studies exist, see Rech et al. [2009], and the MDA official website.

The most important common factor among these approaches is that all of them use standardised notations to create models, then to generate code from those models. Also, patterns are actively used by all these approaches within the context of models and the corresponding generators to aid in the process of development. For instance, $\textit{X}UML$ is very helpful in domain specific modelling. Within a software system, there are several domains and sub-domains each performing a different function. For example, service domains are used for security and networking, application domains provide the context to run the actual application, architecture domains deal with the architectural decisions and so on. The abstraction of the domain is not dependent on the underlying implementation details, and $\textit{X}UML$ can be used to create different abstract models for each of these domains with the help of elements like class diagrams and domain charts, Mellor et al. [2002]. According to Fowler [1997], under the $\textit{X}UML$ approach, UML behaves like a normal compilable programming language.

Furthermore, in each of the model driven approaches, styles/patterns form an integral part and are used extensively. Also, Domain Specific Modelling Languages DSLs are used for model development.

As mentioned earlier, MDSD is an effective software development approach that speeds up the software development process. It creates compact, precise, high quality models which vary between semi-formal and formal, based on stakeholders needs. The created models have a direct mapping with the code which is generated using them, and form an integral part of the software system. The focus of the created models is to solve the domain related problems. Hence, the choice of programming language is unimportant, thereby allowing higher levels of abstraction. This holds well, not only for MDSD but also for other model driven approaches irrespective of the domain, Stahl et al. [2006]. Some of the main advantages of the MDSD approach are described in Table 2.5 and it can be seen that MDSD provides various advantages in terms of cost, time and effort, irrespective of the domain and technology. However, identifying domain and models types is a very important aspect of MDSD and will be explored further in the next section.

<table>
<thead>
<tr>
<th>Table 2.5: Advantages of MDSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspects</td>
</tr>
<tr>
<td>MDSD properties</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Focus is on domain knowledge as compared to technical knowledge, since domain knowledge is more important from the functionality perspective. Technical knowledge is integrated into MDSD platforms by experts so that it can be used by developers during application development. Due to the ready availability of technical knowledge, fewer experts are needed for technology consultations. Business knowledge is captured by models and technical knowledge is captured by platforms. Due to these integrations, knowledge repository is available during the entire project life cycle.

<table>
<thead>
<tr>
<th>Expert knowledge, availability and usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development and implementation time is reduced.</td>
</tr>
<tr>
<td>Newer technologies can be adopted through automation.</td>
</tr>
<tr>
<td>Due to automation, speed is increased and requirement for manual intervention reduced.</td>
</tr>
<tr>
<td>Marketing time is less.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software development automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall quality is increased.</td>
</tr>
<tr>
<td>The non-functional QA related to infrastructure is clearly segregated from functional code thereby making it easier to adopt new technologies.</td>
</tr>
<tr>
<td>Due to automation, the number of bugs is reduced and hence the maintenance of systems is much easier, in terms of technology and architecture.</td>
</tr>
<tr>
<td>Maintenance and rework is reduced and hence client satisfaction is high.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Creating high quality applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since technology is captured by platforms and transformations, it is not tightly bound to the application and hence changes to technology do not reflect in application model changes.</td>
</tr>
<tr>
<td>Every time a new technology is adopted, it needs to be mapped to the platform once, then it will be available for future use.</td>
</tr>
<tr>
<td>which will increase business value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extensive decoupling of technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any changes or enhancements are captured via models thereby saving the application and infrastructure from disturbances.</td>
</tr>
<tr>
<td>Since the software specifications and design are not dependent on the technology, the model is more robust and consistent irrespective of underlying details.</td>
</tr>
</tbody>
</table>

There are several levels of model maturity that may apply during model development. Each of these levels has a set of characteristics with respect to the specifications and approaches.

According to the model development level descriptions proposed by Rensink et al. [2006], I have mapped those levels into the formality levels presented in Table 2.3, which help to visualise and compare both characteristics, in order to help precisely set the target of the notations formality and models maturity during architecture description, based on the business needs, as illustrated in Table 2.6.

Regarding software patterns, they are a means of solving software problems which occur repeatedly in different scenarios. The solutions for such problems involve the creation of good architecture and/or design descriptions by skilled and knowledgeable architects and designers. When recurring problems can be solved with the help of these designs, they become known as best practices which eventually evolve into established patterns. Patterns are most useful at
higher abstract levels. They can be used in one single abstract layer (for example, architecture patterns, or implementation patterns) or they can span across multiple abstract layers (for example, patterns which involve a combination of architecture and design, such as Layer or Pipe-Filter styles). The MDSD approach uses patterns extensively and the application of patterns can be automated. Whenever newer elements are created, the existing elements are also modified in such a way as to remain in-sync with the patterns used. For example, in an architecture using the Pipe-Filter family, whenever a new element is created and attached to the family using ACME-Studio (as the modelling tool), this new element conforms to the existing patterns which have been used within the family. This pattern conformance feature is an advantage of this approach. MDSD is a powerful modelling approach which has a variety of features and characteristics.

More information and brief summary of MDA advantages and disadvantages is illustrated in two figures, in Appendix B and Section B.2.

2.4.1.1 Importance of domains in MDSD

Any system can be described by a combination of domains, and domain identification is a crucial part of the MDSD approach. A domain refers to a particular knowledge spectrum with boundaries. It is an independent unit made up of several relevant elements which also belong to that particular area of knowledge. The entities or elements of a domain depend on the existence of other entities in the same domain, but are independent of the entities’ existence in other (external) domains, unless it is linked to that other domain. However, any domain can use the functionalities of other domains and can similarly provide its own functionalities for use by other domains. Also, domains could be real or imaginary and each domain has a set of affected behaviours associated with it, Mellor et al. [2002]. For example, the surveillance domain (Radars) within Command, Control, and Communication (C³) systems provides functionalities to the (Weapons) domain, like target information (speed, coordinates, etc.), through the logic domain (C³) main processor.

The same situation applies in C⁴I systems, where, for example, Air Operations Centre (AOC) as (a mission-planning domain) publishes its missions to database (another domain).
This database is accessible by Air Tasking Order (ATO)/Airspace Control Order (ACO) (as another domain) that encompasses thousands of sorties/missions and a very large amount of related information. The Operational Unit (as another domain) subscribes to ATOs. An ATO can be edited so that the Unit views only those missions assigned to it.

Before embarking on an application development life cycle, it is important to identify and segregate the domains that will make up the system as a whole. Generally, software systems are made up of the application domain (e.g. an e-commerce site, such as Ebay), technology domain consisting of the technologies used in development (e.g. Java and J2ee), database domain consisting of the database providers (e.g. SQL and Oracle) and any other middle layer domains which may be relevant to the application, like message frameworks, user interfaces and business processes, considering the fact that sometimes architecture is considered as a domain by itself.

While identifying the domains, it is necessary to define clear functional boundaries so that each domain represents a specific purpose through the entities making up that domain. For example, an e-commerce site (e.g. Ebay) represents an application residing in a domain and made up of entities like orders, payments and rules. Similarly, a graphical user interface has entities, such as windows and forms, with functionalities deciding the application behaviour on actions like click and hover. Domain partitioning enables the segregation of application from underlying implementation details thereby allowing creation of layered models. Once the domains and sub-domains have been identified, they can be represented through models using modelling techniques. It is useful to divide the domain into sub-domains, because smaller units provide better control. A system can be divided into one or more sub-domains. Generally, sub-domains fall under the following 2 categories:

1. Technical sub-domains – These are portions of a system which are identified based on a technical aspect and need to be modelled based on a specific language. These are contained within the parent domain, (e.g. graphical user interfaces and persistence).
2. Partitions – Domains are often divided into several partitions to enable parallel functionalities or load-balancing. For example, within an insurance domain several product types, like Vehicle, Buildings and Life, can be defined. These partitions can further be decomposed into smaller parts, such as the Vehicle being decomposed into Motor Vehicle and Marine Vehicle, Stahl et al. [2006].

Since domains are an integral part of MDSD executions, it is important and necessary to identify the domains belonging to a system before attempting to model and formalise them. Domain segregation is also important to identify the inputs for automation processes.

2.4.2 Introduction to MDA

Model Driven Development (MDD), proposed by the Object Management Group (OMG) in 2002, is a fast evolving field Kleppe et al. [2003b], which provides a platform to develop software applications. It provides assistance to the model driven software development process.
2.4. MODEL DRIVEN APPROACHES AND ARCHITECTURE

The basis for MDA is the creation of high-level abstract models. Generally, there is a slight ambiguity in the programming community about the classification of MDSD and MDA concepts. Some of this ambiguity originates from the fact that MDSD efforts without OMG are sometimes referred to as MDA. In this context, Fowler’s description outlining the difference between the two seems more accurate. According to Fowler [1997], MDA can be classified as a specific version of MDSD\(^5\), which uses the standards of OMG.

2.4.2.1 MDA Framework

There are several major participants, as mentioned below, which together make up the MDA framework:

1. Models – These are further divided into Platform-independent model (PIM), and Platform-specific model (PSM).
2. Languages – These include the languages in which models are created, the transformation definition languages and the meta-languages.
3. Transformations – This also includes tools with which to perform transformations.

Figure 2.9 illustrates a basic diagrammatic representation of the MDA concepts and Figure 2.10 highlights the differences between basic, extended, and complete frameworks of MDA.

---

\(^5\)In this thesis, all software modelling methods discussed are laying under the umbrella of the MDSD.
CHAPTER 2. BACKGROUND

Metalanguage

Transformation definition language + Extended MDA Elements Form Complete MDA

Extends

Is written in

Transformation definition language

Is written in

Metalanguage + Basic Elements Form Extended MDA

Figure 2.10: The basic, extended, and complete MDA framework, Kleppe et al. [2003a].

1. Models are used to represent the system of interest and can be of two types:

   - Models that are independent from the platform in which the system is going to be implemented, known as Platform-independent model (PIM)s.
   - Models which are aimed at specific platforms and will not work with other platforms, known as Platform-specific model (PSM)s. The PIMs are usually created using modelling languages like UML/SysML. PSMs can be created in different languages depending upon the needs specified, as illustrated in Figure 2.9.

2. A model is described using a well-defined language. Models in one language can be converted into models in another language using transformation definitions.

3. Transformations from one model to another are done using transformation languages which are well-formed languages working with a set of transformation tools. The tools read the instructions, written in a transformation language, as definitions and execute them at the Meta-level.

4. The transformation definition languages and the modelling languages either use or extend meta-languages. Meta-model languages and Transformation languages are important parts of MDA, Kleppe et al. [2003a].

   It is seen that the aim of MDA is to separate the platform details from the business logic. This essentially means that business logic, design and specifications should be platform-independent and should behave in the same manner irrespective of the underlying technologies. This ensures that changes in technology do not hamper the application behaviour and also that the business knowledge is not bound to the technical details, and remains common across platforms. All this
ultimately means adhering to the separation of concern concept, and that models comprise the core of the system and different platform-specific codes can be generated as long as the models are available.

In the Model Driven Architecture approach, developers usually concern themselves with the development of platform independent models. These models are then converted to platform specific models using tools, which are able to understand the definitions. These operations are performed in stages sequentially. After creation of the PSMs, they are further transformed into code. The entire process can be defined in one line as illustrated in Figure 2.11.

![MDA basic process](image)

Figure 2.11: MDA basic process.

These factors indicate that the life cycle of MDA follows traditional life cycles, like waterfall, wherein the steps are carried out sequentially and each step in the sequence produces an output which can be used as input for the next stage. The output models are usually machine-understandable just like traditional models except for the fact that the conversion from one model to another is done with the help of tools as opposed to the traditional manual processes. Semantically, all these models are well-defined using modelling languages with the aid of software patterns and styles. The majority of these models are bound with Object Oriented (OO) concepts and methods. The association of OO methods with MDA concepts is explored in the next section.

### 2.4.3 Object Oriented (OO)-method

The Object Oriented approach is well-established in the field of software development. The OO-method in the context of model driven architecture and development process is based on the idea of creating models that satisfy the OO-concept, which represents the business requirements accurately in order to obtain a high quality product. The main schema of OO-methodology with respect to model driven approach are as follows:

1. The modelling principles have to satisfy Object Oriented concepts and have to be defined in an accurate manner.
2. The traditional OO concept can be integrated with the formal and latest modelling concepts to create a high quality framework.
3. All the architectural layers and views represented by the models have to be represented accurately in the finished product and this can be done by maximising automation of code with the help of code generators. The layers may include static and dynamic aspects as well as the front-end.

The OO-concept merged with model-driven development well. Martin et al. [1994]. Its concept of agility is often used in the Agile methodology with its integrated platform, Qumer et
al. [2008]. This ensures rapid development to create user-friendly code satisfying current market needs. The two main phases of OO-based model driven development are: a) Conceptual modelling representing the problem; and b) Code Generation representing the solution. Figure 2.12 shows the OO-Method concept.

Figure 2.12: OO – method concept schema, Pastor et al. [2007].

The following models are created by using the OO-method for MDSD, including some examples from my experience with C3 systems:

1. Object Model – This represents the structure of objects and instances known as classes belonging to the domain under consideration. This also represents the relationships between different objects and their service activators. For example, in C3 system, targets and consoles are object classes which can interact with each other. Whereas, a specific target with ID, is considered as an instance.

2. Dynamic Model – This represents the behaviour of the objects identified and the order in which interactions and events occur within the system. For example, a target is followed by identity creation.

3. Functional Model – This deals with the changes occurring in the system when the state of an object changes due to some action. For example, an enemy target disappears when destroyed.

4. Presentation Model – This deals with the concepts related to the presentation of the system in terms of views and user interfaces. The views are responsible for the way the user interface behaves and also for capturing the user interactions. For example, in a C3 system, the airspace tactical maps is a view belonging to the presentation layer.

2.4.3.1 Key aspects of OO-method with respect to SA, SPs, and QAs

The main features of Object Oriented methods with respect to SPs and QAs are described below:

1. Any approach following the OO-method must ensure that all the information related to architectural and structural SPs present in the system must be captured and recorded accurately so that the correct components can be created using modelling methods. Also, the approach should capture the patterns in behaviour in order to enable the creation of proper expressive models. Therefore, these patterns act as the inputs for the creation of models.

2. After capturing the architectural information in the system, the architecture has to be doc-
2.4. MODEL DRIVEN APPROACHES AND ARCHITECTURE

umented in a precise order and the important patterns must be marked.

3. The use of OO Method for a specific SP that is related to a specific QA, will have an affect on the overall system’s qualities, as explained in Chapter 3.

According to Pastor et al. [2007], patterns in the architecture are an important part of the product creation. These patterns must be studied and used well so that a high quality product is obtained. However, there are some problems associated with this approach.

1. First and foremost, there is a shortage of empirical data that supports the relationship between architecture/design Patterns and QAs. Hence, obtaining measurable quality using these patterns is still a difficult task; although there is plenty of research that has been done in this area, such as the analysis of architecture styles workshop conducted by Kim et al. [2006], Bass et al. [2013], Mistrik et al. [2014].
2. Second, the quality and efficiency of the patterns are dependent on the environment and quality measurements. What may work well in one environment may not hold good for others. This leads to inconsistent results.
3. Third, sometimes patterns interact with each other through interfaces and it is not clear if the quality attributes of a particular pattern will apply to that pattern alone or be valid across interacting patterns as the implementation details may vary for each pattern.

2.4.3.2 Mapping OO-Proceses to MDA-Proceses

From the previous sections, it can be seen that OO-development phases are most similar to MDA-development phases, in order to produce high quality softwares with the help of transformation mechanisms and generators. This section will explore the relationship between the OO-method and MDA in order to examine the similarities between them and also the extent to which patterns are applied within their models.

The OO driven modelling approach is made up of 2 phases:

1. Conceptual Modelling – This involves the identification and creation of platform independent models known as PIMs in MDA approach.
2. Code Generation – This involves the generation of code from the models using transformation techniques, which includes two steps as follows:

   • Initially, the PIMs are converted into platform specific models or PSMs.
   • then the PSMs are converted into source code for the application. The transformations define the mapping between PIM to PSM and PSM to code.

OO-method phases can be directly mapped to MDA stages, Pastor et al. [2007], as shown by Table 2.7.
Table 2.7: Analogies between MDA and the OO-method.

<table>
<thead>
<tr>
<th>MDA</th>
<th>OO-Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform-independent model (PIM)</td>
<td>Conceptual Model</td>
</tr>
<tr>
<td>Platform-specific model (PSM)</td>
<td>Application Model</td>
</tr>
<tr>
<td>Implementation Model (IM)</td>
<td>Application Code</td>
</tr>
<tr>
<td>PIM-to-PSM transformation</td>
<td>Mappings</td>
</tr>
<tr>
<td>PSM-to-IM transformation</td>
<td>Transformations</td>
</tr>
</tbody>
</table>

2.4.3.3 Extra features of OO-methods missing from MDA

There are certain features in the OO-methodology that cannot be directly mapped to MDA. According to the Kleppe et al. [2003b], PIMs may often contain all the information necessary to generate the application code, thereby eliminating the need for any extra information. Whereas, the OO-method is more definite due to the fact that the Object-Oriented Administrative Systems-development in Incremental Steps (OASIS) specification language provides a very precise platform which means that the OO-models contain complete information needed for code generation each and every time as opposed to sometimes.

Another difference between the two approaches is that in the case of MDA, whenever the PIM contains all the necessary information, the PSM becomes obsolete and is no longer required. In such cases, the code can be generated directly from the PIM, Pastor et al. [2007]. On the other hand, in the case of OO-method modelling, it is necessary to define an application model which is platform specific and this cannot be bypassed even though it is not mandatory for the application model to be revealed. This difference goes to prove that the MDA PIMs contain some level of platform details required for code generation which means that they are not platform independent. Whereas, conceptual models generated by the OO-method are completely platform independent. This implies that the OO-method models can be run on any platform accurately.

Overall, this means that the models generated using the OO-method are complete entities by themselves which contain all the computational information necessary for code generation and it is possible to create complete systems with these models without the need for any other predefined elements, such as libraries and controls, Pastor et al. [2007].

2.4.4 eXecutable and Translatable UML (\(X^T\)UML) as an example of model automation approach

eXecutable and Translatable UML (\(X^T\)UML) is one of the main contributors towards MDA and helps in generating models which meet the system architectural specifications, Figure 2.13 illustrates the \(X^T\)UML concept.

Executable UMLs can be considered as extensions or supplements to the traditional way for defining system specifications in terms of its interfaces and constraints. The traditional definitions containing these elements use the Interface Description Languages (IDL), and the PIMs generated by \(X^T\)UML can be mapped to these definitions easily. From \(X^T\UML’s point
of view, the platform specific models are not mandatory, unlike PIM, and can be discarded as XTUML considers PSM to be just an intermediate layer between PIM and code, Mellor et al. [2002].

Executable UML focuses on domain-specific modelling that creates models, based on the domains. The domains are generally identified based on the requirements wherein the requirements are broken down into several use cases and each use case is put under a specific category representing a particular subject area. All the use cases belonging to a particular category together form a domain. Domains and their use cases are mutually dependent on each other. Domain experts may have to go back and forth to arrive at exact specifications related to the two. Once the domains and use cases are identified, XTUML can be used to create high level abstract models of the domains, and the domains combined together form the big picture of the system architecture.

Irrespective of the differences between the various modelling approaches, such as (MDA and OO-methods), patterns are used uniformly within the models and transformation engines; hence, patterns play an extremely important role in all approaches. As a result, modelling methods benefits and drawbacks will be reflecting in the overall architecture.

2.4.5 **Key aspects of MDSD approaches with respect to SA, SPs, and QAs**

Model Driven Software Development (MDSD) does not have a clear-cut relationship with SA. However, there are certain relationships between the two, which are described further in this section.
2.4.5.1 Models and transformations

There are two possible transformations that can be identified with respect to model driven approaches. They are known as *Horizontal Transformation* and *Vertical Transformation* Punter et al. [2008], which are illustrated in Figure 2.14.

The identification of transformation depends on the direction in which it is happening. If it is happening from a higher-level to a lower-level (or vice versa), it is known as **Vertical transformation**; for example, transformation from PIM to PSM or PSM to Code. Even reverse engineering code to obtain the high level models is considered as vertical transformation. Contrary to this, **Horizontal transformation** occurs when the transformation happens at the same level. In the case of horizontal transformations, the transforming is usually between models at the same level; i.e. between two abstract models of the same hierarchical level. Typical examples of this include code refactoring and code refinement, where one model is transformed into another at the same level based on the business needs.

Irrespective of the type of transformation, there are a few quality parameters present across all of them. Table 2.8 summarises some of the **MDSD factors** that have an effect on different QAs, from the point of view of three types of transformation techniques, Rech et al. [2009]:

1. Horizontal transformation approach – proposed by Röttger et al. [2004], uses partial automation techniques (to transform context models) to provide good response times in models by refining them.
2. Horizontal approach – proposed by Merilinna et al. [2004], uses complete automation techniques (to transform architectural models), in order to provide better performance and reliability in models by refactoring them. *This approach is good to use during the evaluation of architectural facts.*
3. Vertical approach – proposed by Kurtev [2005], uses complete automation (e.g. for UML models) to provide better adaptable models by synthesising them.
<table>
<thead>
<tr>
<th>Proposal</th>
<th>Purpose</th>
<th>Type of</th>
<th>Input artefact</th>
<th>Quality attributes</th>
<th>Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zou <em>et al.</em> [2003]</td>
<td>Reverse engineering (migration)</td>
<td>Vertical (CM-to-PIM)</td>
<td>Program Code</td>
<td>Coupling and cohesion</td>
<td>No</td>
</tr>
<tr>
<td>Röttger <em>et al.</em> [2004]</td>
<td>Refinement</td>
<td>Horizontal</td>
<td>Context models</td>
<td>Response time</td>
<td>Partial</td>
</tr>
<tr>
<td>Merilinna [2005]</td>
<td>Refactoring</td>
<td>Horizontal (PIM-to-PIM)</td>
<td>Architectural models</td>
<td>Performance, availability, reliability</td>
<td>Yes</td>
</tr>
<tr>
<td>Kurtev [2005]</td>
<td>Synthesis</td>
<td>Vertical (PIM-to-PIM)</td>
<td>UML class models</td>
<td>Adaptability</td>
<td>Yes (Mistral)</td>
</tr>
<tr>
<td>Markovic <em>et al.</em> [2005]</td>
<td>Refactoring</td>
<td>Horizontal (PIM-to-PIM)</td>
<td>UML class models</td>
<td>Syntactical correctness</td>
<td>No</td>
</tr>
<tr>
<td>Sottet <em>et al.</em> [2006]</td>
<td>–</td>
<td>–</td>
<td>Interface models</td>
<td>Compatibility, error protection, homogeneity-consistency</td>
<td>No</td>
</tr>
<tr>
<td>Kerhervé <em>et al.</em> [2006]</td>
<td>Synthesis, refinement</td>
<td>Horizontal and Vertical</td>
<td>Information models</td>
<td>Response time, network delay, network bandwidth</td>
<td>No</td>
</tr>
</tbody>
</table>
2.4. MODEL DRIVEN APPROACHES AND ARCHITECTURE

Patterns and Transformation:

Each of the techniques mentioned in Table 2.8 takes an input model and converts it into an output model, except for the first approach by Zou et al. [2003], which reverses the code to get models. The transformation process is accomplished with the help of supporting transformation tools and techniques.

Some of the tools and engines used for this purpose make use of certain patterns and styles to achieve the transformation. Figure 2.15 illustrates the relationships between models - models, and models - real world elements (domains), while Figure 2.16 illustrates the implementation of the observer pattern (as an example), which shows the transformation from one model to another, along with the dependency between GUIs and entities.

![Figure 2.15: Relationship between models and real world.](image)

Sometimes, the output models may have properties which are different from the input models. In those cases, patterns may be used to create the output models with precision and desired quality. For example, while converting general domain models (e.g. architectural models) to specific models (e.g. quality model), the reusability metrics must be maintained, as discussed by Becker [2008] and Moreno et al. [2008] performance model in Section 2.5.3.6. Also the structures in the models corresponding to the solution structure provided by a particular pattern are created using transformations. There are several aspects or forces which affect the applicability of a pattern, such as its qualities, Schumacher et al. [2006]. Also, there are several variations of each pattern which can be used according to the needs, as each variation often comes with some advantages and drawbacks. The user of the pattern has to study these and take the design decisions accordingly, Stahl et al. [2006].

Utilising MDSD approaches makes it easy to use patterns in some situations by providing code generation features with the help of models. For example, to create manually the dependencies and event registrations along with notifications for the observer pattern in the code is a difficult job. It is much easier to create models which represent these aspects and generate the code from those models. It may be necessary occasionally to customise or extend the pattern.
While using patterns in the model-driven environment, it is important to segregate the application and transformation models since the pattern usage is different for each. The decisions regarding the patterns in transformations must be made by the developer who creates the transformations, and the application developer needs to make decisions about the implementation details of those patterns. Pattern description and documentation issues are discussed in Chapters 3 and 4.

2.4.5.2 Pattern Languages with respect to SA (in brief)

Pattern languages are generally groups of patterns intended to solve a particular problem which might be complex and too minute to be solved by normal means Greenfield et al. [2004]. Some examples of pattern languages are Enterprise Application Architecture Patterns, Fowler et al. [2003], Security Patterns, Schumacher et al. [2006] and Remoting patterns, Greenfield et al. [2004].

These groups of patterns shall usually be well-defined and sequential. Each pattern is dependent upon the preceding pattern and hence the dependencies also should be well-defined. However, these patterns are indicative of the system’s behaviour. They are designed to solve common problems within the system. Hence, it is a good approach to describing, combining, and classifying SPs. On the other hand, the Gang of Four (GoF) and Pattern-Oriented Software
Architectures (POSA) patterns are more generic and it is possible to use any pattern independent of the others.

However, according to Alexander et al. [1977]: “No pattern is an isolated entity. Each pattern can exist … only to the extent that it is supported by other patterns. So, patterns are not individual independent entities and any pattern can exist only when it receives support from other patterns.” Patterns can form much of any architecture, that’s why it is important to this research objective, as can be seen from the analysis of the RCS-reference model in Chapter 6.

The use of UML macro definitions is to create models that are sometimes defined as patterns by certain tools and modelling approaches. However, many dispute this because a pattern is considered to be much more complex, Greenfield et al. [2004]. At the same time, it is agreed that building groups of patterns which address some generic problem makes their usage and integration much easier with systems. For example, the Enterprise Data Model built by Silverston et al. [2009] using patterns, aims to solve the problems related to the enterprise architecture and database domains.

The strengths and weaknesses of these kinds of approaches are described below:

1. Strengths – They provide a good balance between creation of general patterns to solve specific needs. They are consistent and easy to integrate.
2. Weaknesses – It is difficult to arrive at a common modelling style as there are several levels and layers of patterns involved. Also, it is difficult to understand the rules regarding pattern usage, because several models with multiple levels of details are combined together.

Despite the weaknesses of some of these approaches, the idea of pattern languages is still useful when it comes to modelling and automation. According to Schmidt et al. [2000] and Greenfield et al. [2004], the main features that should be present in a pattern language are:

1. The language should provide patterns which deal with all aspects of software development, such as architecture specification, architecture refining and system implementation.
2. The patterns should be created according to some common schema so that all the patterns in the family are consistent with each other and can be searched and compared easily.
3. All the relationships between a family of patterns, such as dependencies, associations, extensions and containment, should be well-defined and known.
4. The structure should be easy to navigate and to provide alternate options.
5. Implementation guidelines should be well-defined and documented so that the patterns can be easily implemented within applications.
6. The patterns should expose information about their structure, rules, etc., so that they can be modified easily if needed.

Pattern languages are useful for the creation of meta-model elements, which are needed in the creation of Domain Specific Language (DSL). For example, in the case of a Remoting pattern implementation in order to generate remoting infrastructure elements, pattern languages
are useful to identify the elements that need to be represented by DSL. Some of these elements are interfaces, invoker, request handlers for client and server, pooling, leasing elements for Life-Cycles, and communication elements, such as callbacks and poll objects. For all these elements, the pattern languages can be useful in identifying the configurable factors for each. Other information concerning the elements, such as their structure and their relationships, can also be encompassed in a meta-model using DSL.

All this proves the usefulness of patterns with respect to the model driven approach. However, according to Lange et al. [2005] and Rech et al. [2009], quality concerns related to individual patterns and merging of patterns, interfacing between patterns, etc., still remain unclear. Chapters 3 and 4 of this thesis will explore some of these concerns.

- **Pattern Family is a subset of Pattern Language:**

According to Alexander [1979], patterns in problems often lead to the discovery of solution patterns. Also, pattern families help in finding reusable generic solutions to several domain related problems irrespective of the field and sub-domains Greenfield et al. [2004]. As an example, the Remoting Patterns Völter et al. [2004] and the Security patterns Schumacher et al. [2006] are patterns that combine to work together inside the same domain, whereas in case of the Garland family of patterns (described by Shaw et al. [1996]) every family is considered as a single pattern with several components such as the case in the Client-Server family. Each family is made up of three elements: Property types, Constraints, and Structures.

According to Schmidt et al. [2000] (POSA-V2), many of the popular patterns are merely used for the creation of frameworks and hence those frameworks can be considered as concrete implementations of the abstract patterns. In fact, most of the efficient frameworks are made up of several pattern implementations, Greenfield et al. [2004]. Similar to the pattern families, the modelling languages can also be combined together to form a blueprint for domain specific software development. Due to these features, modelling languages are considered to be more efficient and feature-rich as compared to normal programming languages. With this in mind, it might be a good idea to consider the generation of frameworks as pattern families. A group of patterns working together to achieve a common goal can be implemented together to create a framework with automation techniques. For example, distributed applications often use web services and web methods. Patterns can be used to implement these requirements and a combined framework may be generated to be utilised by the distributed applications on various platforms.

Studies have also shown that as the level of abstraction increases, the contribution towards solving a problem decreases Greenfield et al. [2004]. However, the decrease in patterns utilisation to solve problems, is affected by other factors other than abstraction levels, as described in Chapter 4. Keeping this in mind during pattern creation and implementation, the scope of the problem needs to be kept narrow in order to increase the applicability of the patterns. Most of the earlier patterns mentioned in the GoF Book, and POSA-V1 are quite broad and generic.
2.4. MODEL DRIVEN APPROACHES AND ARCHITECTURE

in nature, while most of the newer patterns are more specific to a particular problem domain. For example, security patterns deal with the problems associated with the security domain. The patterns defined by POSA-V2 have been categorised as pattern languages and pattern systems, as explained in Chapter 3. However, the informal details provided for these patterns, and several other patterns as well, are not sufficient for quality measurements or for automations. Measurement of quality is a notable concern in most cases.

- Weaving patterns into languages:

The references available about patterns are not sufficient to explain how to use patterns or to create solutions for problems, Greenfield et al. [2004]. Also, patterns are extremely important to create models and to address quality related issues. The Kim et al. [2006] workshop tried to address some of these issues with the help of the Alloy tool. This workshop also tried to address mapping specifications between architectures and models, while maintaining properties like consistency, reliability, and style compliance.

The patterns development trend in the last two decades is focusing more towards the development of pattern families and languages as opposed to individual patterns, Schmidt et al. [2000], Schumacher et al. [2006], Manolescu et al. [2006], Zimmermann et al. [2008], Schmidt et al. [2013]. Combining several patterns into one language is known as weaving patterns into languages. It has also been predicted that pattern languages are the way forward for solving domain specific concerns and coarse problems, Greenfield et al. [2004].

A pattern language could be considered as a way to define modelling languages. Most of the formal languages are made up of several patterns which are already in use by other languages. For example, languages like C# and Java are made up of properties, events, delegates, etc. The definitions of these patterns can be used by the compiler to generate implementation details. The main difference between the pattern language and modelling language is that the implementations details are exposed in the case of pattern languages whereas they are encapsulated partially or completely in the case of modelling languages.

2.4.6 Conclusion

In the context of Software Architecture (SA), this section explored model driven approaches and their different flavours. It also explored the importance of patterns with respect to the model driven approach in terms of domain models and transformation models. The grouping of patterns into families and languages was also discussed along with their relationship with the modelling methods. To conclude this section, it has been found that the MDSD and SPs are important technologies that could help in improving the field of SA description and evaluation.
2.5 Evaluating Software Architecture

A number of studies have critically analysed software architecture evaluation methods that proposed several feature-based criteria in order to assess and compare these methods, and to evaluate different characteristics of Software Architecture (SA), such as Abowd et al. [1997], Dobrica et al. [2002b], Clements et al. [2002a], Obbink et al. [2002], Babar et al. [2004], Kazman et al. [2005], Mårtensson [2006], Qin et al. [2008], and Athar et al. [2016]. In addition, others have also made an effort to report architecture evaluation’s best practices, such as Best Practice.

In this section, the architecture evaluation in general will be discussed, followed by analysis of some specific evaluation techniques in regard to their assessment of Software Architecture (SA) models, as well as the complexity and tradeoff between different quality attributes. Also, analysis of some related works that tackle the relationships between patterns and quality attributes will be discussed. It will be then concluded by explaining QAs within the context of SA.

2.5.1 Evaluating software architecture in general

Software architecture evaluation has been recently emerging as an important area of research and practice resulting in the active development of methods, techniques and tools by researchers and practitioners, Overhage et al. [2007]. System architecture is also considered as a centre of development process, which has a big impact on other development phases. This is why architects are generally suited to be the leaders of almost every development team and why architecture is one of the top careers in the system development arena.

Two basic classes of evaluation methodologies were defined for architecture evaluation about twenty years ago namely measuring and questioning, Abowd et al. [1997]. Measuring employs quantitative metrics by defining precisely the numerical scale to the targets. Therefore, only those attributes that are easily mapped to quantitative metrics, e.g. response time, can serve as inputs for this kind of technique throughout the link in the network. Metrics, simulation, prototype, and specific experience are required to perform this type of evaluation. The second methodology, Questioning, which provides questions to check qualitative attributes which expands its suitability to almost any given quality attribute. This class includes scenarios, questionnaires, and checklists. Questioning techniques are primarily used for most current architecture evaluation and are commonly supplemented by Measuring.

Throughout the architecture, design, implementation, and deployment phases, one should always consider the achievement of QAs. It is a matter of getting the big picture (architecture), and the details (implementation) correct to obtain satisfactory results.

Some of the techniques that have been used to evaluate SA are briefly explained in the next section.
2.5. EVALUATING SOFTWARE ARCHITECTURE

2.5.2 Comparisons between current common architecture evaluation approaches

Many different approaches and methods have been introduced to evaluate the SA in the software community.

In order to understand some of these specific methods, the basic knowledge of the software evaluation methods must be understood. In this section, a brief explanation and comparison among some of those approaches will be discussed. Figure 2.17, illustrates the broad evaluation domains. Most, if not all, existing evaluation methods and techniques fall under one of these domains.

Figure 2.17: Evaluation techniques.

- **Questionnaires and Checklists:**
  
  A questionnaire, which applies to all architectures, is a list of wide-ranging, reasonably open questions.

  A checklist, is a detailed set of questions developed after much experience in assessing a common set of systems, frequently domain-specific. Checklists tend to be much more focused on particular functions or quality attributes of the system.

  Questioning techniques are the broadest category of review techniques. Unlike measuring techniques which actually require the existence of some artefact to measure, such as the architectural models in this case, **Questioning techniques** include questionnaires and checklists which are all used to elicit a discussion about the architecture and to better understand the fitness of the architecture based on its given requirements. Also, they may be used to investigate any area of development phases-(virtually any state of readiness).
• **Scenarios and scenario-based methods:**

A *scenario* is a short description of an interaction with the system from the point of view of one of its stakeholders. Scenarios are usually specific to the system whose architecture is being evaluated. An example of a scenario-based evaluation technique is the Survivable Network Analysis (SNA) method developed by the SEI Coordination Centre. Permission of survivability properties’ methodical assessment of proposed systems, existing systems, and modifications to existing systems is SNA’s main objective.

• **Measuring techniques:**

Unlike questioning techniques, *Measuring techniques* are used to answer specific questions about specific quality attributes. They require the presence of an architecture, a design, or implementation artefacts. A review based on measuring techniques needs to focus not only on the results of the measurement, but also on the assumptions under which the measurement was deemed useful. For example, assumptions about resource utilisation patterns are made based on a calculation of performance characteristics, Clements *et al.* [2002a].

• **Hybrid techniques:** Methods that are combined elements from two or more evaluation methods, unlike those previously discussed, which fall clearly into one of the three camps. Hybrid techniques may be combined Questionnaires and Checklists, Scenarios, and Measuring metrics, Qin *et al.* [2008]. An example of such a hybrid method is Software Performance Engineering (SPE).

In the next two sections, we briefly discuss the scenario-based methods and measuring techniques, due to its importance to this thesis objectives and approach. Whereas, other methods are the most known methods to the readers, and they were explained sufficiently within this section.

### 2.5.2.1 Scenario-based evaluation methods

Architecture evaluation methods are used to assess SA, where the specified quality attribute should be collected during both requirement gathering and evaluation phases, which then is represented by an initial or a series of architecture description candidates. Since quality attributes have different characteristics, defining the measurement of these attributes is essential before evaluating. This ensures that the attribute can characterise the capability of software in meeting the requirements.

The scenario-based method is the most notable of all architecture evaluation methods. It uses scenarios or hypothesised sets of the system’s uses or modifications Dobrica *et al.* [2002a]. A scenario covers a wide range of possible behaviours that may be done to the final system. In the context of architecture evaluation, scenarios are used to represent a concrete quality attribute. Some of the most popular evaluation scenario methods are discussed with critique as follows.
2.5. EVALUATING SOFTWARE ARCHITECTURE

2.5.2.1.1 Software Architecture Analysis Method (SAAM) was created in 1993 and published in 1994. It is the first well-documented and carefully designed analysis method for architecture analysis Kazman et al. [1994]. During the early 1990s, when software architecture was still not widely accepted, it was already a remarkable effort, Qin et al. [2008]. Improvements were made later by getting more detailed descriptions and the financial managements system’s study cases by (Kazman - 1996), which have been cited by Bass et al. [1998]). SAAM is a simple, easy to learn and carry-out method that requires minimal amount of training and preparation.

SAAM is intuitive and simple; intuitive because it uses scenarios instead of quality attribute description to measure a software’s quality; and simple because it only considers the relationship between a scenario and architecture structure. Some evaluation methods, such as ATAM, were introduced after SAAM due to its control over many common quality attributes. SAAM also prepares a platform for stakeholders where they can discuss their ideas and concerns about the system’s blueprint and to resolve the understanding of deviations and incorrect architectures and/or designs. The general phases of evaluation, achievements for each phase, and the relationship between them comprise SAAM’s steps. The primary steps are shown in Figure 2.18.

SAAM however has one pitfall since it does not provide a clear quality metric for the architectural characteristics or attributes that are being analysed. Also, SAAM does not involve the use of tools, because there are no tools to support theSAAM evaluation method until recently. There are a few tools such as the Morale tool with limited support for SAAM, Zayaraz [2010].

2.5.2.1.2 Architecture Tradeoff Analysis Method (ATAM) is considered as SAAM’s advanced version. Aside from SAAM’s capabilities, ATAM also helps to better understand the trade-off to multiple relatives and even inconsistencies of quality requirements or targets. While
most experts were trying to enhance SAAM, its investors took notice of the relationship between targets reflected by scenarios and their effects on system construction.

ATAM is built upon three areas: architectural styles; quality attribute analysis, and, SAAM, Qin et al. [2008]. It could be regarded as a hybrid technique because it uses questioning, scenarios, and measuring techniques. Questioning is based on architectural styles, quality attributes, and pre-existing questions. Scenario, is due to its main use of general and specific scenarios. Measuring by using quantitative outputs from reliability, performance and security models, Clements et al. [2002a].

ATAM’s main component consists of four phases that comprises of nine steps, in total. The preparation before the first phase and follow-up after an evaluation finish, can be considered as another two phases, which are not included in ATAM main phases. ATAM phases are represented in Figure 2.19:

**Each phase contains different steps as follows:**

- **Phase 1**: Presentation – exchanging information through presentations, (3-Steps).
- **Phase 2**: Investigation and analysis – assessing key quality attribute requirements with respect to architectural approaches, (2-Steps).
- **Phase 3**: Testing – checking the results to date against the needs of all relevant stakeholders, (3-Steps).
- **Phase 4**: Reporting – presenting the results of the ATAM, (1-Step).

*More explication for ATAM steps as follow:*

- **Presentation**
1. Present the ATAM – Evaluation methods are described to the participants, expectations are set and questions are answered by the evaluation leader.
2. Present the business drivers – Business goals motivating the development effort and what the primary architectural drivers will be, are described by the project spokesperson (ideally the project manager or system customer).
3. Present the architecture – Architecture is described by the architect, focusing on how it addresses the business drivers. In this step, I think that the explanations and justifications of the architecture choices by an architect is not enough and needs to be accompanied by an evaluation mechanism and results for the architecture candidates, to be able to support the chosen architecture. Its this reason, one of many that encourage me to do this research.

• **Investigation and analysis**

4. Identify the architectural approaches – Architectural approaches are identified but not analysed by the architect.
5. Generating the quality attribute utility tree; eliciting and specifying the level of scenarios. Annotating with stimuli, responses, and prioritising the quality attributes that comprise system utility. For this step, the employing of the tactics that were introduced by Bass et al. [2013], could be a very helpful approach.
6. Analyse the architectural approaches – Architectural approaches that address high-priority scenarios from Step 5 are elicited and analysed. Identifying architectural risks, non-risks, sensitivity points, and trade-off points are done in within this step.

• **Testing**

7. Brainstorm and prioritise scenarios – From the entire group of stakeholders, a larger set of scenarios is elicited, which are prioritised via a voting process involving all the stakeholders.
8. Analyse the architectural approaches – This step reiterates the activities of Step 6 but uses the highly ranked scenarios from Step 7. These scenarios are the test cases to confirm the analysis performed. Additional architectural approaches, risks, non-risks, sensitivity points, and trade-off points may be uncovered and documented through this analysis.

• **Reporting**

9. Present the result – Findings based upon the information collected during the ATAM evaluation is presented to the assembled stakeholders by the ATAM team. Figure 2.20, shows the steps of the ATAM based on SEI description, Clements et al. [2002a].

ATAM was upgraded and enhanced in 1999 after being applied to several practical projects.
There are two major improvements of ATAM to be observed.

First, is the concern on how to realise when it is suitable to discontinue the generation of scenarios. To address this concern, a set of quality attribute-specific questions has been developed by SEI, by which one can find that some useful scenarios are still missing and try to supplement them. More details about the questions may be found in SEI’s website\(^6\).

Second, is the adoption of Attribute-Based Architectural Styles (ABAS). ABAS, is a type of analysis-assistant tool, which helps the stakeholders to identify quality attributes brought by architectural styles. Since ABAS is a combination of stimuli, responses, and architectural decisions based on attributes and an analysis models, thus it may be described as an architectural style/pattern attached by attribute values, to reflect quality information. However, in the case of performance as an example, relevant information is not enough. An analytic framework will be used to facilitate analysis from the information gathered, Qin et al. [2008].

\(ABAS\)^7 is composed of four parts:

1. Problem description – a description of the problem that the structure solves.
2. Stimuli/responses – a characterisation and description of the stimuli that ABAS is designed to respond to, as well as a description of the quality-attribute-specific measures of the response.
3. Architectural style – relevant components to the quality attribute such as the set of component and connector types, the topology, a description of the patterns of data and control interaction among the components, and any properties of the components or connectors.

\(^6\) www.sei.cmu.edu
\(^7\) More information about ABAS can be found on SEI’s website. See footnote 6.
2.5. EVALUATING SOFTWARE ARCHITECTURE

4. Analysis – a quality-attribute-specific model that provides a method for reasoning about the behaviour of component types that interact in the defined pattern.

Since ATAM was created, it has experienced a continuous evolution and improvement. Some initial materials may be found in the works of Kazman et al. [1998] and Bass et al. [1998], while ATAM’s further detailed study cases can be found in Bass et al. [2013] and Clements [2003] works. The latest status of ATAM can be found on SEI’s website including the tutorials and support material. Figure 2.21 shows the latest conceptual flow of ATAM.

Figure 2.21: A conceptual flow of ATAM, SEI [2010].

2.5.2.1.3 Active Reviews for Intermediate Design (ARID) is an evaluation method for partial architectures. It is situated between the intersection of two approaches. The first is scenario-based design review techniques, such as ATAM or SAAM that were discussed earlier. The second is active design reviews or ADRs. ARID is best suited for evaluating a partial architecture/design in its formative stages. Figure 2.22 illustrates the two phases of this evaluation method.

Active Design Reviews (ADR)

ADRs are an effective technique for ensuring quality detailed designs in software. This method actively engages reviewers using carefully structured tasks, which avoids asking Yes/No questions. Reviews may be undermined by such questions, which enabling the reviewer to answer the questions without much consideration. In contrast, a sequence of exercises in this method that based on test concrete and not feigned understanding are asked by ADR to reviewers to utilise design.

Evaluating detailed designs of logical units for a software is the primary use of ADRs. Examples are modules and components. Its questions tend to address two things as illustrated below:
1. The quality and completeness of the documentation.
2. If the design’s provide sufficiency, fitness, and suitability concerning the required service.

There are many other SA evaluating methods, which are mostly scenarios-based, each one does have its own unique focus and process such as:

- Architecture Level Modifiability Analysis (ALMA).
- SAAM founded on Complex Scenarios (SAAMCS).
- SAAM by Integration in the domain (ESAAMI).
- SAAM Evolution and Reusability (SAAMER).
- Architecture Level Prediction of Software Maintenance (ALPSM).

An overview of the above methods as follow:

- **ALMA method:**

  The outcome of combining existing architecture level modifiability analysis approaches with scenario-based SA analysis approaches, which focus solely on modifiability is the Architecture Level Modifiability Analysis (ALMA) method. ALMA method comprises five stages as illustrated in Figure 2.23; bear in mind, performing these steps is done with iteration, Bengtsson et al. [2004].

Kazman et al. [2005] did analyze ATAM and ALMA methods in terms of fifteen criteria; and they suggested several ways to improve both methods. Discussing these criteria is outside the scope of this research. However, there are some drawbacks for the ALMA methods such as:

- limits the attributes under inspection to modifiability.
- provides slight tractability support form the goal throughout the analysis outcome.
- in general the authors interpretations for the results do not follow any specific techniques.

More study and experiments could improve the ALMA method.
2.5. EVALUATING SOFTWARE ARCHITECTURE

- **SAAMCS, ESAAMI, and SAAMER methods:**

  SAAMCS, ESAAMI, and SAAMER as extended methods of SAAM. Where SAAMCS is focused on flexibility, Lassing et al. [1999]. The goal of SAAMCS is to use complex scenarios for risk assessment. On the other hand, ESAAMI and SAAMER developed by Molter et al. [1999] and Lung et al. [1997] respectively, are designed for reusability, Dogru [2010]. In one hand, ESAAMI is an integration of SAAM within a specific domain with a reuse-based development process, which reuses the knowledge defined by SA’s models. However, ESAAMI cannot evaluate SA considering more than one QA at the same time. On the other hand, SAAMER provides a framework that collects stakeholders, scenarios, artefacts, SA, and QAs information, in order to support system reusability and evolution, Kim et al. [2010], Alebrahim [2017].

- **ALPSM method:**

  ALPSM method’s main goal is to predict the size of change during maintenance using series of scenarios. Its focus on maintenance effort, defines a maintenance profile, and does not check if the architecture fulfill any other QA, Qin et al. [2008]. ALPSM consists of six steps as illustrated in Figure 2.24, Bengtsson et al. [2004]. However, ALPSM method does not consider reusability of the existing knowledge base, Dobrica et al. [2002b], Alebrahim [2017].

![Figure 2.23: The ALMA method five steps.](image)

**Figure 2.23: The ALMA method five steps.**

**Figure 2.24: The ALPSM method six steps.**

However, it is evident that most of the mature architecture evaluation methods, such as ATAM and SAAM are using qualitative methods that are normally applied through scenarios. It makes the resulting evaluation heavily skewed on the basis of selected scenarios and the interpretations given to required quality attributes. The generation of these scenarios is solely based on the vision of the stakeholders, Clements et al. [2002a] and Zayaraz [2010].
2.5.2.2 Measuring techniques

The main focus of measuring techniques is the demands to know the QAs emanating from the architectural structures. The results derived from these techniques are usually represented in the form of numerical values. They are generated in response to quality issues raised by the various stakeholders. These techniques are only able to answer a few questions revolving around quality attributes such as performance and modifiability. Due to the various complexity issues which surround measuring techniques, understandability matters often arise amongst stakeholders. Examples of measuring techniques include simulations, metrics, prototypes, and experiments, Zayaraz [2010].

The following subsections explain important concepts related to measuring methods:

2.5.2.2.1 Metrics: is “a system of weights and measures to express a hundred times the unit” Simpson et al. [1989]. Also, it is defined as “a system of weights and measures based on the principle that each quantity should have one unit”, Walker [1988].

On a particular observable measurement of the architecture, the placed quantitative interpretation is called metrics. Fan in/fan out of components is an example of metrics. Answers to overall complexity that suggest locations that likely have to change or where change might be difficult, can be provided by the most well-researched measuring techniques.

The following discussion represents a set of metrics (as an example), in order to measure complexity and therefore predict areas of change in a real-time telecommunications system built, using object-oriented design (Arora, 1995), cited in Clements et al. [2002a]). As opposed to code, examining a detailed design can produce more information which makes the following metrics more appealing.

- Number of component clusters – An object that composes the number of component clusters it contains. For example, a computer is composed of a screen, keyboard, software, etc., and other objects,
- Depth of structure: An object that is defined by the number of layers of encapsulation.
- Number of events: An object that reacts to a number of synchronous and asynchronous calls.
- Number of synchronous calls: The total number of synchronous calls from one object to other objects, either to get or set some data/resource.
- Number of asynchronous calls: The total number of asynchronous calls from an object to other objects.
- Depth of Finite State Machine (FSM): Measures the depth of an indirection where an object’s behaviour and the states of this behaviour are described by FSM.
- Number of data classes: The total number of data classes an object uses or refers to.
- Number of extended state variables: The number of variables an object’s FSM needs to
2.5. EVALUATING SOFTWARE ARCHITECTURE

deal with the machine’s synchronisation aspects.

• Depth of inheritance tree: An object’s total depth from the base class in the system’s inheritance tree.

The source of faults is also predicted by other metrics. For example, below are two sets of some metrics that were shown to be useful indicators of fault-prone modules in the telecommunication domain (Khoshgoftaar -1996), which have been discussed in Clements et al. [2002a].

Call graph metrics

• Modules used: This module uses a number of modules, directly or indirectly, including itself.

• Total calls to other modules: This is the number of calls to entry points in other modules.

• Unique calls to other modules: This module calls a number of unique entry points in other modules.

Control-flow graph metrics

• If-then conditional arcs: This is the number of arcs that are not loops and contain a predicate of a control structure.

• Loops: This is the number of arcs that contain a predicate of a loop construct.

• Span of conditional arcs: Within the span of conditional arcs, a total number of arcs is located.

• Span of loops: The number of arcs plus the number of vertices within loop control structure spans.

More of the SA metrics can be found in Bass et al. [2013], Kan [2003], Jaquith [2007], Ejiogu [1991], Ejiogu [2005], and Carroll et al. [2007].

2.5.2.2.2 Simulations, prototypes and experiments: Creating and clarifying the architecture may be achieved by building a prototype or a simulation of the system, but creating a detailed one for review purposes is usually expensive. A portion of a normal development process typically contains these artefacts; therefore using these artefacts during a review or in answering questions encountered during the review, becomes a typical and natural process.

The answer to an issue raised by a questioning technique, could be solved by using simulation or prototype. Thus, if the review team asks “Can you support this assertion with any evidence?”, one convincing answer would be the result of a simulation. This questioning technique is useful in the pattern case. For example, What evidence do we have that proves Pooling patterns support performance?

2.5.2.2.3 Automated tools and architecture description languages: Representing architecture through the use of formal/semi-formal notations and languages has been common during the last two decades. One of the most popular utilised Architectural semi-formal notations and is
the Unified Modelling Language (UML), Merilinna et al. [2004], Milicev [2009], and Silingas et al. [2011].

Emerging from the academic and industrial research communities are several architecture description languages, such as ADLs (discussed earlier in Chapter 2). Creating, maintaining, and analysing architectures are dependant upon the language and tools maturity. If ADL can describe the behaviour of the system then it most certainly has the tool environment to generate a simulation of the system. Simulations like this provide early insight into architecture inaccuracies that lead to behavioural errors and reveal performance bottlenecks. Also, many of these tool environments can turn architectural specifications into executable source code by using the right tools, if available.

2.5.2.2.4 Software Performance Engineering (SPE): To see whether a system as designed will meet its performance constraints, architecture could be examined through the use of a hybrid analysis technique called SPE. SPE is not constrained in a Yes or No result. It is so much more than that. Its purpose is to help the architects and/or designer illuminate and navigate among the trade-offs that are available to them. The intent is to build in performance rather than add it to the system’s architecture / design by scheduling SPE evaluations early and often.

Table 2.9: Comparisons between different evaluation techniques, after Clements et al. 2002a.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Quality Attribute(s) covered</th>
<th>Approach(es) Used</th>
<th>When applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questionnaires</td>
<td>Various</td>
<td>Predefined domain specific questions</td>
<td>Can be used to prompt architect to take certain design approaches, or any time thereafter.</td>
</tr>
<tr>
<td>and checklists</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario-based</td>
<td>Various; either non-run-time attributes such as modifiability or run-time attributes such as security.</td>
<td>System-specific scenarios to articulate specific quality attribute requirements; scenario walk-through to establish system’s response.</td>
<td>When architecture design is complete enough to allow scenario walk-through.</td>
</tr>
<tr>
<td>methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAAM</td>
<td>Modifiability, functionality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARID</td>
<td>Suitability of design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNA</td>
<td>Security</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metrics</td>
<td>Various; often emphasise modifiability and reliability.</td>
<td>Static analysis of structure.</td>
<td>After architecture has been designed.</td>
</tr>
</tbody>
</table>
2.5. EVALUATING SOFTWARE ARCHITECTURE

<table>
<thead>
<tr>
<th>Simulations, prototypes, experiments</th>
<th>Various; often emphasise performance, functionality, usability.</th>
<th>Measurement of the execution of an artefact.</th>
<th>After architecture has been designed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMA</td>
<td>Performance oriented to real-time systems.</td>
<td>Quantitative static analysis.</td>
<td>After the process model has been built and process-to-processor allocations have been done.</td>
</tr>
<tr>
<td>ADLs</td>
<td>Various; tend to concentrate on behaviour and performance.</td>
<td>Simulation, symbolic execution.</td>
<td>When architectural specifications are complete.</td>
</tr>
<tr>
<td>Hybrid Techniques</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPE</td>
<td>Performance.</td>
<td>Scenarios and quantitative static analysis.</td>
<td>When performance constraints have been assigned to architectural components.</td>
</tr>
<tr>
<td>ATAM</td>
<td>Not oriented to any particular quality attributes, but historically emphasises modifiability, security, reliability, and performance.</td>
<td>Utility trees and brainstormed scenarios to articulate quality attribute requirements; analysis of architectural approaches to identify sensitivities, trade-off points, and risks.</td>
<td>After the architecture design approaches have been chosen.</td>
</tr>
</tbody>
</table>

Different common Software Architecture Evaluation (SAE) approaches have been briefly discussed, each with its own capabilities. It is also explained that each evaluation method has its own use. Table 2.9 illustrates the comparisons between different SAE approaches.

Most of the above evaluation methods mentioned above can contribute in building analytical models, which could involve in the new software model-driven development approaches, automation checks with formal scenarios, and measuring techniques, in order to explore the qualities within the architecture or software patterns.

In the next section, some specific evaluation methods that contribute to the SAE in particular will be discussed.

2.5.3 Analysis of Specific Evaluation Techniques

Over the past 20 years, there has been an increase in recognition to achieve quality attributes such as dependability, performance and security. It is critical for the satisfaction of system’s requirements that architects/designers must pay careful attention to their architecture/design. Because of this recognition, the field of software architecture has significantly matured in this period. However, relatively informal methods have been the basis of recent software architecture
practices. This limits the potential to acquire insight and enhance the quality of the resulting system by fully taking advantage of architectural designs, Garlan et al. [2006a].

Styles and patterns provide a domain-specific design vocabulary, and a set of limitations on how that vocabulary may be used. Many architectures and designs use this vocabulary (e.g. Styles and Patterns), as their backbone. Although, numerous tools are available to help for individual architecture analysis. It is unfortunate that architects/designers acquire little or no help for choosing the proper styles/patterns to achieve the desired QAs. The reason is, that only few works have been done in developing such tools, I have introduced a partial solution to this problem in Chapter 3.

Although, there are many approaches as well as researches that analyse specific quality attributes and its relationship to the architecture in general, and styles in particular; whereas, other research analyses the quality attributes for a specific domains, such as communications or distributed systems. It is evident that most of the research areas are concerned on performance and reliability which are very important qualities. However, the importance of these quality attributes in the system depends on the system’s context and business goals. Therefore, a good approach should be able to look into different quality attributes and analyse them with respect to the system’s requirements.

In order to clarify and analyse important and individualised thinking towards SAE, a critique analysis of some specific SAE approaches will be discussed

2.5.3.1 Software Engineering Institute (SEI) – Bass approach

Throughout the three versions of Software Engineering Institute (SEI) tactics, Bass et al. [1998], Bass et al. [2003], Bass et al. [2013]; their tactics concept has demonstrated evolution. They introduced the relationship between architecture and quality attributes in the form of tactics. “A tactic is a design decision that influences the control of a quality attribute response.” A collection of tactics is called an architectural strategy or architectural pattern package tactics. This approach is considered as a scenario based model.

• What important architectural decisions influence the achievement of quality attribute requirements?
• To what stimuli must the architecture respond?
• By what criteria is the achievement of a quality attribute is measured?

Based on the above questions and arguments SEI-group, introduced the tactics methods. Bass, amongst others, discusses the quality attribute decision known as tactics. Individual quality tactics will not be discussed here but the general approach will be described, alongside with the performance tactics as an example

A model is provided by the SEI-team to describe a scenario. To normalise various scenarios into a standard form, six elements are adopted. This facilitates later evaluation processes.
2.5. EVALUATING SOFTWARE ARCHITECTURE

The Bass-team\(^8\) scenario representation model is illustrated by Figure 2.25, while Figure 2.26 explains how the model works in performance case.

Figure 2.25: Quality attribute scenario representation model by Bass et al. [2013].

Figure 2.26: Performance characteristics: stimuli, responses and architecture decisions, Klein et al. [1999].

- Source of stimulus: The stimulus is generated by an entity (a human, a computer system, or any other actuator).
- Stimulus: When arriving at a system, the condition that needs to be considered is called a stimulus.
- Environment: Only within certain conditions can a stimulus occur. The stimulus can take place while the system is running or in an overload condition as long as a condition is true.
- Artefact: An artefact is stimulated; this may be the whole system or just certain components of it.
- Response: After the arrival of stimulus, an activity commences. This is called response.
- Response measure – A response, when taken place, should be measurable in some manner in order to test the requirement.

Redundancy is an example of tactics that have been used for achieving availability, which reveals two things:

1. Tactics can refine other tactics – Bass-approach acknowledged redundancy as a tactic.

\(^8\)For the rest of this thesis (SEI and Bass) means same team and method.
embedded control system) can be considered as its refinement, which are also tactics. To make each type of redundancy more definite, a designer may use further refinements.

2. Patterns package tactics – Both redundancy tactic and synchronisation tactic may likely be used by a pattern which supports availability. More concrete versions of these tactics will also likely be used by this pattern.

It is important to point out that, although Bass-method organised the tactics for each quality attribute as a hierarchy, this hierarchy is meant only to explain some of the tactics, and any list of tactics is not essentially complete. For each of the six attributes that he elaborated, which are: (availability, modifiability, performance, security, testability, and usability), they discussed tactical approaches for achieving each one of them. A brief discussion and an organisation of the tactics were presented by the SEI-team. In order to locate an appropriate tactic, the architect uses the path that is provided by the tactic tree. This approach was able to determine several scenarios, namely, General Scenarios and Concrete Scenarios. General Scenarios are independent of the system. They are based in and can refer to any system, in general. On the other hand, Concrete Scenarios are specific for a given system under consideration. Also, he presented attribute characterisations as a collection of general scenarios, which are will be employed into the proposed initial conceptual framework in the future work, as explained briefly in Section 7.4.

Unlike checklist techniques, using scenarios are system-specific and are not limited by a particular domain, which is an advantage. More so, multiple stakeholders’ suggestions can be synchronised in this scenario. Given the instance that similar cases are interpreted differently by stakeholders, a redundancy elimination process can merge these perspectives. Examples of these evaluation methods include SAAM, ATAM, ARID, and others as described in the previous sections, Qin et al. [2008].

This approach is commendable because it links software quality and its architecture including styles through the use of some tactics. It is also to be understood that achieving quality attributes by identifying some tactics included in the styles helps developers to choose the right styles for their intended system’s goal. However, developers, especially (architects), need to identify all possible tactics for each quality attribute and try to understand the styles and map these tactics according to the right style. This is difficult and time-consuming but it could succeed if the developer understands the style/pattern and its implementation choices.

Not to forget that, implementation is an important concern and that can give the developer different choices about tactics. One solution that could be helpful for developers in their pattern and quality selection, is to prove the relationship between these two aspects and to build an incremental database that summarises all the proven relationships between patterns and qualities. As I explained this solution in Chapter 3.

Comparison between the quality models of SEI that are discussed above and the ISO/IEC9126 model shows some differences. The selection of ISO/IEC9126 model for comparison was in 2010 before the completion of new standard (ISO-25010).
2.5. EVALUATING SOFTWARE ARCHITECTURE

However, the differences between both ISO standards above and SEI lies in the sub-characteristics and attributes that are involved, e.g. availability assessment, as the combination of maturity with fault tolerance, alongside recoverability, Bass et al. [2013]. Bass presents architectural tactics that are related to one or more attributes. Consequently, ISO/IEC 9126 provides more support to this prototyping, based on experimental architectural prototyping. This can be attributed to the fact that quality assessment in this prototype is based directly on metrics. Bass’ assessments are based on quality attribute scenarios that require a need for metrics in order to make a decision on whether a response measure agrees with specifics, Reussner et al. [2005].

In general, the software architecture in practice SEI approach is one step toward exploring the relations between architecture and quality attributes. This approach needs more investigation on other possible tactics, formalising their strategies to achieve the desired qualities through these tactics, and most importantly, to prove that the selected style or pattern support required quality.

The general aim of this research is to improve SAE. To achieve this, the architecture determined by a set of elements, interaction mechanism, semantic constraint, implementation structure, and topological layout that need to be noted. Although all these aspects affect the quality attributes, these attributes are difficult to tackle using this approach. On the other hand, formalising this approach and mapping these tactics with more intensive strategies to the architectural artefacts and pattern’s languages could be the best way to help evaluate models and automate this relationship. For example, using advance use case modelling for discovering, identifying and modelling the problem context, including the quality requirements for a system and the creation of proper scenarios will help in building the relationship between architectural models and qualities in systematic ways, Armour et al. [2001]. Figure 2.27 provides an example of Bass tactics for performance with more elaboration.

After the process model has been built and process-to-processor allocations have been done, the performance tactics illustrated in Figure 2.27 could utilise Rate Monotonic Analysis (RMA) method to evaluate performance, which is oriented to the real-time systems. This is considered as a quantitative static analysis approach.

Rate Monotonic Analysis (RMA):

Ensuring that a set of fixed-priority processes is scheduled on a CPU, so that no process ever misses its execution deadline is a quantitative technique called RMA. It is a way to assure that a system will meet its real-time performance requirements. RMA can be performed in the absence of an implementation, which is a powerful advantage. Only a set of process definitions with timing and synchronisation information is required, Klein et al. [2012].

RMA’s application is straightforward. Analysis will remain valid and the system will meet its performance goals as long as the implementation does not violate the constraints given in the concurrency model and that no process runs too long or introduces new synchronisation constraints not accounted in the model. To make sure that no process misses its deadline, an
algorithm that computes the ability to schedule a process must produce an affirmation. This
algorithm is made available by characterising the context of a real-time system in predefined
terms and by specifying the number and nature of the processes to be scheduled. If the algorithm
returns otherwise, then the design will have to be modified.

Performance (Basic contributors to the response time)

- Blocked Time
- Resource Consumption

Performance tactics

- Resource Demand
- Resource Management
- Resource Arbitration

- Increase Computation Efficiency
- Reduce Computational Overhead
- Manage Event Rate
- Control Frequency of Sampling
- Introduce Concurrency
- Maintain Multiple Copies
- Increase Available Resources
- Scheduling Policy

FIFO
- Dynamic Priority Scheduling
- Fixed Priority
- Static Scheduling

Earliest Deadline First (EDF)
- Semantics Importance
- Deadline Importance
- Rate Monotonic

Contention for resources
Availability of resources
Dependency on other computations

Figure 2.27: Performance tactics with elaboration, Bass et al. [2013].

2.5.3.2 Satisfying QAs through the use of SPs, Babar et al. [2005]

The approach used by Babar et al. [2005] is done by systematically analysing existing
security patterns in order to identify those patterns that have architectural implications to sup-
port the design and evaluation of security sensitive architectures. To understand the mechanism,
these patterns are provided to achieve security. The most common interpretations of security are
grouped into small sets of security attributes (as tactics) based on extensive studying and iden-
tification. A relationship between security and software architecture is established by analysing
several security patterns (as a pattern language), in order to study their effect on the identified
security attributes, then the identified security attributes are decomposed into more detailed el-
ements and properties. Figure 2.28 identifies the relationships between their security attributes,
security patterns, and security properties, shows their framework that relates to the problem and
solution domains for the security attribute. While figure 2.29 shows Auditability Flow (as an
example), from problem domain into implementation.
Furthermore, in the tractability Table 2.10, each security attributes (as a problem area) have been examined throughout security properties to discover out if they do have security patterns as (a solution). Two-problem areas, as shown in the table have linked to main properties, but did not linked to any solution in Babar et al. [2005] tactics model. One justification could be that the main properties satisfied the required security QA by utilising sub-properties.

However, the list below extracted from Figure 2.28, shows all sub-properties that are un-
Table 2.10: Absence of solutions to some problems in Babar et al. [2005] model.

<table>
<thead>
<tr>
<th>Security Attributes (Problem)</th>
<th>Security Properties</th>
<th>Security Patterns (Solution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability → Error Management</td>
<td>Error Management</td>
<td>No solution linked</td>
</tr>
<tr>
<td>Maintainability → Simplicity</td>
<td>Simplicity</td>
<td>No solution linked</td>
</tr>
</tbody>
</table>

linked to both sides (problem side and solution side), while their parents are linked to one or both sides, which raise the question about their usability in Babar et al. [2005] model.

1. Error prevention.
2. Fallback procedure.
3. Encapsulation.
4. Global information sharing.
5. User interface consistency.
7. Access verification.
8. Data protection
9. Reducing exposure to attack.

2.5.3.3 Mapping between SEI Tactics approach and Babar et al. [2005] Pattern approach

The achievement of high quality systems that satisfy all stakeholders, is the goal of both approaches (SEI team and Babar et al. [2005]).

Based on the descriptions of both models and with insightful analysis and comparison, the mapping chart between both approaches in Figure 2.30 represents some of the similarities between both models.

Besides the nine sub-properties listed in Section 2.5.3.2 above that still persist in this case, Table 2.11, shows new security attributes, properties, and patterns that have not been utilised by or mapped to SEI model.

Table 2.11: Non-mapped elements between Babar et al. [2005] model and SEI model.

<table>
<thead>
<tr>
<th>Security Attributes (Problem)</th>
<th>Security Properties</th>
<th>Security Patterns (Solution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Error Management (Error Handling)</td>
<td>Full view with Errors</td>
</tr>
</tbody>
</table>

According to Bass et al. [2013], tactics that have been produced in software architecture are not enough and not limited. This can explain some missing links in Figure 2.30. For example, the reliability from Babar et al. [2005] quality attributes list does not linked to any of the SEI tactics. The same applied on the Bass Intrusion Detection tactic, which is not linked to any of Babar et al. [2005] attributes or properties.

Whilst, Intrusion Detection tactic could be linked to one or more of Babar et al. [2005] properties such as Global Information Sharing property to share a network intrusion detection profiles through knowledge-base, which could store packets involved in known attacks, Kruegel
2.5. EVALUATING SOFTWARE ARCHITECTURE

et al. [2005]. Also, the same tactic could be linked to Security Policy property to set up security policy if intrusion happens.

However, systematic evaluation to prove the relations between the selected patterns and selected quality attributes in both approaches has not been analysed in scientific methodology Freitas [2009]. Due to the increasing scale and complexity of today’s systems and due to economic considerations, there is a rising need to apply component-based methods in the construction of systems. Consequently, new component-based evaluation methods are required, which allow reasoning about the quality of a system architecture and existing usable software styles, Grunske [2007].

2.5.3.4 Exploring quality attributes using architectural prototyping Bardram et al. [2005]

This approach is to explore architectural aspect through its prototypes from an analytical standpoint. This explanation study presents two well-established quality frameworks, which are the SEI tactics framework, and ISO-9126 framework.

They are learning vehicles for architects when envisioning and learning new architectural constructs, patterns, and ideas. The prototyping approach, presents an opportunity to observe conflicting qualities’ as it is not possible to implement a code that only expose a single quality attributes. Another advantage for prototyping is that they point out to important characteristics the architecture that set them apart from most other evaluation techniques that either formalise (e.g. ISO approach) or simulate (e.g. Bass approach) the architecture.

Also, architectural prototypes have been treated primarily as an evaluation technique, which forces both frameworks above to be grounded in the reality of concrete implementation, and to overlooking architectural implications. Consequently, reducing risks, due to prototyping artifacts closeness from implementation. This method goal, is not to propose architectural prototypes as a replacement for other techniques; rather, its to present the technique as a valuable assessment tool for evaluation. It is important to note that the assessment aspect of this approach is somewhat different from most other evaluation techniques.
CHAPTER 2. BACKGROUND

Reliability
Authentication
Authorization
Integrity
Confidentiality
Audibility
Maintainability
Availability

Error Management
- Error Prevention
- Error Handling
- Failback Procedure
- Failure Logging
- Simplicity
- Encapsulation
- Initialization process
- Security policy
- Low-level security
- Permission management
- Global information sharing
- User interface consistency
- Guidance
- Access control
- User identification
- Access verification
- Least privilege
- Privacy promotion
- Defense in depth
- Data verification
- Data protection
- Private communication/information protection
- Reducing exposure to attack

Figure 2.30: Similarity between Bass et al. [2013] and Babar et al. [2005] tactics.
2.5. EVALUATING SOFTWARE ARCHITECTURE

Architectural prototyping is about building the architecture, not reasoning about it, formalizing it, or simulating it. In this way architectural prototyping is especially well-suited to explore and experiment with architectural tactics produced by SEI-team. It is worth mentioning that ACME-Studio and Armani tools are considered as a prototype environment, Leavens et al. [2000]. Tactics are considered as concrete techniques. Patterns, and ideas may enforce the presence of a required quality attribute in the architecture. Architectural prototypes in essence explore the validity of a given tactic (or experiment with a set of them) to actually provide the needed quality.

Taking into account, prototyping is not that easy, if one means prototyping that can be evaluated by machines. This is because it is almost a draft version of the real product and missing important aspects or requirements in the architecture prototype could mean revision of all the architecture artefacts from scratch, Reussner et al. [2005].

2.5.3.5 Garlan approaches

One of the most important research and related work associated with the scope of this thesis is Garlan’s work regarding architecture and styles evaluation. Garlan and some other researchers performed different analysis research for architecture and styles in particular, using various tools such as ACME-Studio and Alloy. which will be discussed as follows:

2.5.3.5.1 Architecture-driven modelling and analysis study A formal approach to software architecture that leads to enhancements in software quality is described by Garlan et al. [2006a]. Improved design clarity, analysis’ support, and assurance that implementations conform to their intended architecture are included in this approach. They choose Component and Connector C&C architectural structure for their research because such structure can directly convey critical properties related to dependability, such as reliability, security, and performance, Garlan et al. [2006a].

Illustration of the flexibility property mechanism’s to facilitate architectural analysis, and the introduction of architectural properties, both are proposed by Garlan et al. [2006a]. How architectural behaviour can be specified is also shown, as well as concepts of architectural style introduced. Providing domain-specific architectural models and using ACME-Studio tool’s ability, to check for conformance of a style is also discussed. ACME does not specify architecture properties’ meaning, and does not provide their analysis with native support Garlan et al. [2006a]. However, through the use of external analysis tools to obtain insight into the architecture and assess its qualities, properties as such may be evaluated. Off the shelf theory and algorithms can be exploited by calculations in most cases. These analyses provide a powerful support to architectural design such as:

• identifying errors in design early in the process by architects;
• architects are assisted in documenting the expected run-time properties of architectural elements; 
• tool support for feedback provision and analysis results comparison.

Some examples are automotive control systems', rate-monotonic analysis, security simulation based on Monte Carlo style, and lining up theory-based analysis for distinguishing server overloads.

This approach illustrates several ways on how formal architectural modelling and analysis addresses issues related to software architecture. Important software architecture issues include:

1. clarifying design intent;
2. detecting design flaws using the support of rich forms of analysis;
3. making trade-offs between service goals and qualities;
4. guaranteeing that implementations are consistent with their intent, using several tools.

Moreover, the addition of structural constraints on architecture is enabled in ACME. It is recognised as an ACME-Studio extension, a GUI that is used in architecture design is independent from any ADL. Instead of handling behaviour specification, ACME uses pre and post conditions that are transformed into their respective executable assertions at runtime. However, the partial validation cannot be made since static analysis is not provided, Moreno et al. [2008].

The Garlan approach mainly confirms implementation correctness and maps this implementation to its architecture. This approach is an excellent step toward style analysis but needs more effort and research. For example, if a developer does not have external tools such as Monte Carlo-style security simulation, he/she could not check the security of the system by ACME-studio. This is the same with other quality attributes. Also, if a specific external simulation is used to evaluate specific quality attributes such as security, it should be ensured that this simulation includes the scenarios needed. This could be solved with quality attribute analysis model and repository components, as proposed in this thesis future work section.

2.5.3.5.2 Analyzing architectural styles with Alloy – Kim et al. [2006] In this study, the steps in mapping an architectural style into a relational model are explained by the researchers. This can automatically be checked for properties such as whether a style is consistent, satisfies some predicate over the architectural structure, (e.g., if two styles are compatible for composition or if one style refines another). The architectural style is expressed formally in an architectural description language. A model generator called a SAT solver-based model checkers (Alloy–Analyzer) is used to inspect critical properties of styles. This approach can also examine some of the structural properties of architectural styles such as consistency and styles overlapping. However, this method does not handle architectural behaviour, dynamic changes to architectural models and expressions over ACME properties, and other quality attributes.

9 Elements include components, connectors, ports, roles, and all ACME elements discussed in section 2.3.4.
2.5. EVALUATING SOFTWARE ARCHITECTURE

2.5.3.6 Model-driven performance analysis Becker [2008] and Moreno et al. [2008]

This approach describes model-driven analysis based on reasoning frameworks. This performance reasoning framework transforms a design to a model with real-time performance properties that can be analysed using different evaluation procedures, such as rate monotonic analysis and simulation. It is described in Figure 2.31. The PACC Starter Kit\(^{10}\) is a development environment that can generate and analyse code from the same models. Such tools are implemented as a package and independently deployed as an Eclipse plug-in, allowing any Eclipse-based tool that models software systems in a parsable design notation to benefit from the framework by exporting their designs to ICM\(^{11}\) components. Some (Eclipse-based tool) examples are, the IBM Rational Software Architect, Eclipse Model Development Tools (MDT), UML 2, and ACME-Studio. To add a performance reasoning framework plug-in as a modelling tool, is by creating a class that implements the method: `AssemblyInstance.translateDesignToIcm(IFile designFile)`.

![Figure 2.31: Performance framework, Moreno et al. [2008].](image)

The models of this method are created with Eclipse Modelling Framework (EMF), and the Object Constraint Language (OCL), which are implemented with the OCL-EMF validation framework. This approach has five main elements beside the architecture description element, which are:

1. Design-to-ICM translation – translates from architecture descriptions to simplified representations (ICM);

\(^{10}\)The PACC Starter Kit is a set of software development tools integrated to provide unbiased confidence in system behaviour predictions. This demonstrates how existing technologies can be used as such.

\(^{11}\)Intermediate Constructive Model (ICM) is a model that serves as an input to the performance reasoning framework, were significant architectural descriptions’ elements concerning performance analysis, are contained in this model.
2. ICM – a simplified version of the system’s original design, that serves as an input to the interpretation;
3. Interpretation – transforms ICM architecture descriptions into analytical models;
4. Analytical (Performance) model – a model that can be analysed by different evaluation techniques included in the performance analysis components; and;
5. Performance analysis – the element that analyses the performance model by analytic and simulation-based predictors based on Rate Monotonic Analysis (RMA) and queuing theory.

The structural model is less generic than the one of ACME (described in Section 2.3.4). Contrary to ACME, the content of a communication point is made explicit in this model. This model has an option to be required only or provided only. Typed input and output arguments are also required as a part of the needed set of communication elements. Any component model that exhibits the same bidirectional communication points is compatible with the structural model. This characteristic can be found in many component models such as Fractal or SOFA. Fractal is an advanced component model and is associated with growing programming and management support, initially proposed by France Telecom and INRIA since 2001. Distributed Systems Research Group at Charles University in Prague developed SOFA, a component system which provides many advance features such as: ADL-based design, provision of transparent distribution of applications, behaviour specification, and verification based on behaviour protocols and different communication styles that are supported by software connectors, Becker [2008].

To make their created designs non-ambiguous, meta-models with strong semantics were used as models, therefore allowing analyses to be performed. The semantics of communication points must be clear to enable analysis tools to manipulate them. According to Moreno et al. [2008] in ACME/Armani, the content of a communication point (called Port explained in Section 2.3.4) is not expressed. The only possibility to add information in a communication point is by defining properties, which have no clear semantics. For example, it is possible to perform Wright behavioural analysis on an ACME description by adding some ACME properties, containing Communication Sequential Processes (CSP) expressions, Brookes et al. [1984], Roscoe et al. [2011]. It is not possible to guarantee that the messages declared to be sent or received structurally, as there is no relationship between these properties and the structure of the architecture.

In the Moreno approach, the behavioural meta-model is strongly associated with the structural meta-model. The sent and received messages expressed in the behavioural model correspond to the argument in the structural model. For example, this approach checks that the sent (resp. received) messages in the behavioural model correspond to the output (resp. input) arguments in the structural model. More generally, Moreno chose to have a richer structural meta-model than ACME in order to be able to check the coherence between the different meta-models; i.e. structural, behavioural, data flow and quality, Becker [2008]. The scope of this
2.5. EVALUATING SOFTWARE ARCHITECTURE

approach has different aspects and concerns such as decreasing design models and the analysis models’ semantic gaps; also, to enable the analysis tools’ utilisation together with various design languages. However, this approach is limited to one quality attribute. Thus, developing an interpretation component for each quality characteristic for the purpose of its evaluation, is a hard and time-consuming job. Also, reducing the component size of the original architecture to the half number in order to evaluate quality attribute (as the case in performance model) is not delicate; and I think it does not actually reflect the real design, which could affect the integrity of the evaluation mechanism.

Other approaches:

Koziolek et al. [2007] used the Palladio Component Model (PCM) to illustrate component-based architectures to include the necessary information for performance prediction. Developers and software architects will find the features of PCM very useful. Component developers have the ability to determine the performance of their components independently from context. Therefore, they will have the ability to enable predictions on third-party performance as well as improving the reusability of their components. PCM, also allows software architects to analyse the performance of their design without requiring a written code. By using the specifications, they can analyse performance using performance specifications that are retrieved from repositories and assemble them to architectures, Rech et al. [2009].

More proposals, such as the intermediate models CSM by Petriu et al. [2004] and KLAPER by Grassi et al. [2005], have also been made. Also, D’Ambrogio (2005) describes a framework to automate the building of performance models from UML design models, Moreno et al. [2008]. UML model based performance modelling and prediction have already been executed by Gilmore et al. [2003]. It incorporates performance information in the state diagrams’ transition labels, Becker [2008].

Many studies have been reported, in order to identify, analyse, evaluate and explain the relationship between quality attributes and architecture or design including their patterns. Each research has its own scope and focus in a specific domain and quality. However, very few have identified the relationship between quality attributes and architecture in general conceptual schema that can be applied for most quality attributes and can automate the analysis result. For further details and additional knowledge in this area, readers are advised to explore different resources.

2.5.3.7 Evaluating SA using Metrics – Zayaraz [2010]

Most software pattern references explain patterns based on the expertise of the authors and their perception, such as Fowler et al. [2003] and Schmidt et al. [2001], whilst others, such as Kim et al. [2006] work, try to evaluate SA upon measurements and metrics.

In the work of Dr. Zayaraz, he did use different useful available methods within his evaluation framework. For example he applied the rules and principles of the Common Software
Measurement International Consortium (COSMIC) – *Full Function Points* with some metrics to measure the basic interaction parameters for some characteristics such as coupling, cohesion, and complexity on different patterns (e.g. Pipes and Filters). Also, he utilised Analytical Hierarchy Process (AHP) for comparisons between different pattern structures for specific quality attributes. His work was a step forward in regard to creating the relationship between SPs and QAs based on scientific fashion and standardised measures, not just an observation or experience of the pattern author. However, this approach has the following limitations:

- *Useful standardised frameworks such as ISO-42010, were not employed during the development of the framework.*
- *Useful and standardised modelling languages such as UML, ADLs, had not been employed during the development of the models.*
  
  Both points above, could make the approach more concrete, easy to understand, modifiable by other researchers, and reusable.
- *Also, the QAs are handled independently, but in the real world they are working together, as a spider net.*

### 2.5.4 QAs in SA Context

In this section I discuss Quality Attribute (QA) standardisation, complexity, challenges, categorisations, sensitivity point, and trade-off in the context of Software Architecture (SA).

#### 2.5.4.1 Complexity of quality attributes

During the 1970s, the software community already showed interest in quality attributes. Various taxonomies and definitions have been published, many of which have their own research and practitioner communities. Software quality is defined by the IEEE standard 1061 (IEEE 1998) as “*the degree to which software possesses a desired combination of attributes.*”

Different schools and standards have defined and categorised QAs. This variations of QAs definitions and categorizations increase the complexity of the quality system.

According to *Bass et al.* [2013], from an architect’s perspective, there are three problems of quality attribute descriptions:

1. Operational definitions for an attribute were not provided. Therefore, saying that a system is modifiable is rather non-significant. Specific systems may be modifiable with respect to one quality attribute but may not be modifiable to another.
2. Some relationships between quality attributes can fall under the same aspect. For example, a system failure can fall under several attributes, such as availability, security or usability, which make QAs categorisation more complex.
3. Each attribute community may consist of its own set of words that are understood by a specific (vocabulary). Also, the same occurrence may be called by different names in each community. For a given occurrence performance may call it “events,” security will call it
“attacks”, availability will call it “failures” while usability will call it “user input.” As all these refer to the same occurrence. This situation is ambiguous and confusing comparatively.

Using quality attribute scenarios to characterise quality attributes to solve the first two problems mentioned above (non-operational definitions and overlapping attribute concerns), is a solution proposed by Bass et al. [2013]. To solve the third problem, a brief discussion of each attribute should be presented. To demonstrate the concepts which are fundamental to the attribute community, one must concentrate on each attribute’s fundamental concerns.

Judging architecture for suitability based on the names of the attributes is not sufficient. Requirement statements, such as:

- High security is required.
- The system shall exhibit acceptable performance.

If not elaborated, statements like these may be interpreted differently and misunderstood. Also, these interpenetrations could be based on different QAs categories and definitions, which existed as explained in Appendix C. Another aspect is how the current literature categorises them (QAs)? Is it based on the context of the problem, the interpretations of the quality attribute name, problem domain, different quality dependency trees, and/or categorizations?

Quality attributes are not absolute quantities. For example, a system should be protected from intrusions therefore a password is set to prevent unauthorised users from using the system. However, it does not have virus protection mechanisms which results in another kind of intrusion. This illustrates that the existence of quality attributes is based on the context of specific goals.

2.5.4.2 Understanding quality attributes

The architecture of non-trivial systems determines its quality attributes. A statement of the quality attribute requirements motivated by key business goals and specification of architecture is a prerequisite of an evaluation. This prerequisite is often unmet. Quality attribute requirements and architecture documentation are commonly incomplete, vague and ambiguous therefore it is necessary to apply two main processes, in order to determine if the evaluation of architectural design decisions address the quality attribute requirements as follows:

1. A precise statement of quality attribute requirements must be elicited and refined
2. A precise statement of architectural design decisions must be elicited and refined.

For the purpose of this thesis, and to achieve (partially) the objectives of the two points above, the following are some suggestions to be applied:

- The evaluation of any quality attribute shall be based on:
  - a known standardised definition for that QA;
  - a clear metric with known unit;
– and identification of the metric input and output.

- The use of clear notations and standardised rich languages, such as (ACME and SysML), to create architectural models, shall help to make an accurate artefact with correct decisions.
- The use of architectural frameworks guidelines such as (IEEE-42010 and DoDAF) for developing a new architecture, shall help during the development process, views, and the related entities, which need to be constructed.

Styles/Patterns could be utilized much better, if they were documented well. However, it is important to have clear and informative characterisations for each quality attribute since architecture evaluation focuses on them. Hence, the quality attribute requirements need to be concrete and measurable or at least observable during evaluation for adherence.

**2.5.4.3 Quality attribute characterisations**

The classification of QAs should follow a clear criteria, standards, and guidelines, in order to help different disciplines to communicate easier. The more clear descriptions as to how the categorization is done, the more the acceptance there will be, for the categorization schema. The advantages of codifying the relationship between architecture and quality attributes are to enhance the design and analysis process; reuse of existing analysis and determine tradeoffs explicitly by engineer; aids in reconfiguring architectures to provide specified levels of a quality attribute; and, to help in the new modelling and automation approaches to apply quality analysis.

According to Bass et al. [2013], every quality attribute has different considerations to be taken into account for its stimuli, responses, and the architectural decisions used by the designers. Few examples are listed below:

**Stimuli**

- Modifiability – a change request
- Performance – the arrival of events at the system
- Availability – a fault occurring in some portion of the system

**Responses**

- Modifiability – person-days or months required to make a requested change.
- Security – intruders breaking into the system and what resources will be accessed.

**Architectural decisions**

- Performance – the allocation of processes to processors and priorities
- Availability – replication and fault detection and failure protocols

The quality categorization table included in the database (Chapter 3) illustrates the differences between quality attributes that are explained by three schools namely, ISO-9126, SEI, and POSA. While, Table 2.12 illustrates another three respected approaches (as examples), with three different views of quality characteristics (ISO/IEC 9126-1991, IEEE Standard 1061-1992, and FURPSI, Krasner [1999] and Grady [1992]).
2.5. EVALUATING SOFTWARE ARCHITECTURE

Table 2.12: Software quality characterisation, Futrell et al. 2002. They are basically based on McCall Model (1977), which was developed based upon three objectives: to specify (Factors), to build (Criteria), and to control (Metrics).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Behaviour</td>
<td>X</td>
<td>X (Time Economy)</td>
<td></td>
</tr>
<tr>
<td>Resource Utilisation</td>
<td>X</td>
<td>X (Resource Economy)</td>
<td></td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adequacy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compliance</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Customisability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Evolvability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Extensiveness</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Interoperability</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Suitability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value/Satisfaction</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Integratability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Compatibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolvability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Expressability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Integrity</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Openness</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Quality of the Parts</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Requirements Enabler</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Special Topics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Maintainability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysability</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Changeability</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Correctability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Expandability</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Stability</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time-Constrained</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Resource-Constrained</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptability</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.12: (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Installability</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Conformance</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Independence</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Software Independence</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replaceability</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Failure Rate</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Maturity</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nondeficiency</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Recoverability</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Supportability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainable</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Reusable</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Support Response</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Testable</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understandability</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Learnability</td>
<td>X</td>
<td></td>
<td>X (Easy to Learn and Use)</td>
</tr>
<tr>
<td>Communicativeness</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Operability</td>
<td>X</td>
<td>X</td>
<td>X (Easy to Operate)</td>
</tr>
</tbody>
</table>

**Key – X: included in the standard**

#### 2.5.4.4 Sensitivity points and trade-off points

According to Bass et al. [2013], the key architectural decisions are sensitivity points and trade-off points. An architectural decision that has elements that are parts of two or more architectural components is called a **sensitivity point**. These components are critical in accomplishing a specific quality attribute response measure. The response measure is sensitive for changing the decision. For example, certain levels of encapsulation determine the length and scale of effort in person-days that is required to maintain a specific system.

When an architect, a designer, or analyst tries to comprehend the achievement of a quality goal, they identify where to focus their attention through the help of sensitivity points.

A **trade-off point** is an architectural decision that affects more than one attribute by some other attributes, and sensitivity points as well. For example, security and performance can be greatly influenced by changing the level of encryption the predicted security can be improved by increasing the level of encryption but processing time can take longer than usual. An example
of the trade-off between quality attributes, illustrated by Table 2.13.

Table 2.13: Set of quality attributes trade-offs, after Kan 2003.

<table>
<thead>
<tr>
<th></th>
<th>Capability</th>
<th>Usability</th>
<th>Performance</th>
<th>Reliability</th>
<th>Installability</th>
<th>Maintainability</th>
<th>Documentation</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>C</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>C</td>
<td>S</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Installability</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>C</td>
<td>C</td>
<td>S</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>C</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>C</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

C – Conflict; S – Support; Blank – No relationship.

It has been apparent that in order to be able to evaluate SA, one should understand QAs, and investigate around current quality systems, in order to be able to utilise the best of the quality arena approaches and thus to avoid their complexity.

2.6 Research challenges

This section will summarise the challenges that face Software Architecture Evaluation (SAE) in general, and this thesis work in particular. The following three subsections uncover three main aspects: 1) the conceptual challenges of this research; 2) the general challenges influencing the architecture description and evaluation; and 3) the standardisation as a common problem in software architecture.

2.6.1 Research conceptual challenges

Formal methods, modelling approaches, evaluation techniques, architecture description languages, architectural styles, and design patterns are different aspects of software development that contribute to software architecture maturity, either totally or partially. In order to understand the automation mechanisms and architecture evaluation, we need to integrate the necessary and useful knowledge as well as describe the proper methods of these development aspects.

“We must now try to find out how we should go about getting a good fit. Where do we find it? What is the characteristic of processes which create fit successfully?” (Alexander [1964]).

It is substantial, one understood the relationships between the aforementioned points, and to formulate necessary aspects of SA. In order to help to evaluate the architecture accurately, and to automate the evaluation process. It is essential but complicated, because each aspect of the topics mentioned above is a large and growing area of research that keeps producing new
CHAPTER 2. BACKGROUND

ideas, approaches and techniques. At the same time, researchers in these areas sometimes do not see out of their area or over their discipline fences. That leaves them separated from others, decreases their opportunity of integrating different approaches from different areas/disciplines, decreases their chances of using existing knowledge, tools and approaches to produce new ideas and to conserve time. Open minds lead to new inventions and help researchers to cope with technological evolutions.

Furthermore, there are many architecture evaluation approaches. However, none of them have produced fully automated evaluation techniques for different quality attributes with adjustment options. Different software architectures consist of different characteristics that could be totally different in their forms, relations and their level of abstraction, which makes architecture difficult to study and to evaluate automatically.

Dragging architecture from its abstraction level to lower levels for evaluation purposes is not a good idea and most times, when that it happens, it loses its advantage of being the software big picture with no unnecessary details, Clements et al. [2002a]. Software architects and developers use patterns in practically in every aspect of the software developmental process.

The following points blend reasons from different fields that enable the SAE process to become more challenging:

1. Formal methods and languages are important in the evaluation and automation process, when using a novel developmental approach. Automation and new developmental approaches, such as Model Driven Architecture (MDA), base processes on machines, compilers, and transformers. A more precise and formal instructive language input is therefore necessary for machines to understand before processing. Natural descriptive languages such as English, are expressive languages. Even-so, they are non-rigorous and non-formal, and they cannot be processed and interpreted by machines correctly. Good examples of natural descriptive languages for software patterns are written in books published by Gamma et al. [1995] and Buschmann et al. [1996]. However, Architecture Description Languages such as the ADL family, still have some level of formalities that are not yet fully developed. Formalisation of models/patterns with architectural modelling and/or description languages is therefore functionally limited pending the full development, such as Garlan et al. [1997] work in (ACME Studio).

2. Model-driven methodologies are one way in which architecture evaluation can be automated. However, these approaches are also currently still in development; understanding and applying them, for the purposes of this study, is another challenge.

3. Some quality attributes cannot be evaluated in a systematic manner within the context of the architecture (e.g., Understandability). This may be addressed in further researches on the subject.

4. Although various tools are available to describe architectures and patterns, map code, and evaluate architecture, these tools are limited in number and also mostly in the growing
stages (e.g., Alloy, ACMEStudio, Armani, ArchE, and Discotect).

5. Multiple terms exist that refer to the same patterns, the descriptions of which include colloquial terms (e.g., some patterns included in Design of Patterns by Gamma et al. [1995] and Pattern-Oriented Software Architecture (POSA-V1) by Buschmann et al. [1996]. This allows interpretations to vary and thus becomes an obstacle in patterns-tracing and documentation. In this case, the issue can be resolved by applying standardised names and formalised descriptions.

On the other hand, regarding architectures in Artificial Intelligence (AI) particularly; there are five challenging points as follow:

1. Variations in terminologies between software engineers and artificial intelligence research groups with regard to architecture, add to the challenges.

2. In AI groups, the lack of definitive research focus on architecture and its evaluation are a big challenge. As a result, developing a common evaluation method that can be used in many different AI systems is difficult. I think it will be a great contribution to AI discipline if future research regarding evaluation methods can be channelled to the AI arena. Furthermore, Selman et al. [1996] presented two sub-problems that that support SA challengings in AI systems:

   • First, mastering architecture and attention and, second, preferences and utility in modelling. The first is primarily involved with where the data regarding state utility originates, whose particular utility is optimised, and what the most practical utility model is when evaluating finite action sequences. In utility model architecture, varying assumptions result in varying qualities (e.g. efficiencies).
   
   • Second is primarily involved in the development of richer attention models, which can turn the analytic machinery of decision-theoretic inference into the task of problem-solving, while applying satisfactory decision-making. In any case, the opportunity exists for the application of these approaches to maximise the general configuration and nature of any architecture, which includes decisions regarding the result-gathering stage.

3. Also, one of the problems in self-adaptive systems (which considered AI systems), when it comes to modelling dimensions is defining models that can represent a wide range of system properties, Selman et al. [1996]. If the models are more precise, they become more effective in supporting decision processes and run-time analysis. This brings the need for classification and enumeration of dimensions used in modelling, in order, to acquire precise models; also to support runtime, reasoning, and decision-making. As a result, achieving self-adaptability and specific quality. This point takes us back to an earlier argument about how important it is to amalgamate different knowledge (modelling approaches, qualities evaluation, formal descriptions, etc.) in order to develop a richer evaluation model.
4. In addition, in these types of intelligent systems, one concern lies in the mapping of requirements onto architecture, which involves the utilisation of patterns that could fulfil the required quality terms. Also, challenges exist in the building of reference architectures for self-adaptive systems that tackles structural control loop arrangements and quality trade-offs.

5. An engineering challenge also exists in the development of evaluation approaches that are able to automatically identify unintended interactions and specific qualities, Cheng et al. [2009]. One solution to the aforementioned sub-problems in AI can be achieved with the automation of architecture evaluation models using standardised languages and mature tools if available. Furthermore, it will be a more beneficial and an intelligent model for architecture evaluation if it can be made to be self-optimising through adaptation of a dynamic architecture context.

2.6.2 Major architecture developments challenges and debates

2.6.2.1 General challenges influencing the architecture evaluation

In 2000, three important areas of software architecture advancement were mentioned by Garlan [2000]. These are: development of architecture description languages and tools, emergence of product line engineering and architectural standards, and codification and dissemination of architectural design expertise. In addition, three factors that impact the development of software architecture have been introduced. It involves changing build versus buy balance (COTS), network-centric computing, and pervasive computing.

Further investigations into factors that influence “industrial practices of software architecture evaluation” were made by Babar et al. [2007]. It was discovered that; technical, socio-political, business, managerial and organisational factors had roles to play. To be a participant in their studies, one had to have a minimum of 5 years experience as a software architecture designer with an average number of 8 years experience of designing and evaluating software architecture in different industrial domains. Empirical evidence from research has confirmed that there were some influential factors to be considered in architecture evaluation.

Most important, the technical factors that could influence this research direction are:

- Quality attributes being evaluated.
- Integration issues between evaluation model and other environments (e.g. systems to evaluate).
- Representing and visualising architectures.
- The languages to be used and the availability of tools, (e.g. SysML language, Artisan tool)

The practice of the product line engineering is still not common. Thus, we need to have a better understanding of the economics, processes, and objects that could help in the use of the product line approach to become successful. The product line approach requires different
2.6. RESEARCH CHALLENGES

development methods, which is an issue. When it comes to a single product approach, the evaluation of the architecture is based on the specifications of the product itself. This has made it easier to build several single products, which are independent from each other. Each architecture is based on the developmental environment and many assumptions, Garlan [2000].

This challenge makes the integration of different products, from different vendors, which can bridge the gaps between products without any mismatch in their architectures very difficult. As a result, the evaluation process of the end product integration (if we can make the integration) becomes harder. The general architecture evaluation method will allow evaluators to adjust the model to the required environment. This will allow them to evaluate any architecture through an automation process, which is not an impossible task, but we are still far from achieving this.

Architecture description language challenges:

Due to the popularity of box and line depictions for describing architecture designs, which employ non-formal notations lead to a number of problems, such as:

- Designs in this situation cannot be analysed to see if they meet the required set of QAs.
- The architectural constraints that may have been observed at the beginning of the design may not be implemented during the system development.

Also, tools support is important:

The use of tools that support architectural languages and modelling techniques (e.g. ACME-Studio by Garlan, Rational Rose by IBM, and Artisan) has been provided to facilitate developers tasks. Utilisation of the tools involves, parsing, displaying, analysing and simulating architectural descriptions. Thus it defines the importance of tool support in architectures and their evaluation.

The availability of good tools helps researchers to do more quality work with less time, also to blend-in some methods, frameworks, languages, and notations into one approach, if necessary; such as Artisan tool, which support several languages and architecture framework, as explained in future work section (Chapter 7).

Some researchers do care about the methodology and notations of a language during their development of any product, neglecting to pay attention to the tools that support their utilised language, which results in:

- Time wasted, so developing architecture description using tool that supports languages and models creation, is much faster.
- Product with no modifiability, especially if the language is not supported by any tool.
- Loss of interest by other researchers, to develop and improve the existing product.

More information about ADLs can be found in Appendix B, Section B.3

At the same time, we should note that there is a difference between what researchers see as desirable and what can be observed from practice. For example, ADL provides tool support,
but there has been some proposals that focus on areas like analysis, refinement, and dynamism, Medvidovic et al. [2000].

Another challenge facing the SA and SAE are the limitations of some languages with no mechanism of integrating them. For example C2, Rapide, Wright and SADL can support architectural analysis. In these areas, ADL has always left some facets unexplored and focused on a particular technique. This has led to the use of ACME as an architecture interchange language. In order to help during the interaction and instill cooperation between different ADLs and tools, thus filling the gaps. SADL and Rapide are the only tools that can provide support to refine architectures across multiple levels of abstraction. SADL requires the mapping of constructs between the abstract and an architectural style, thus making its support limited. Currently, ACME can only provide visuals and conformance of the architectures.

It has been observed that, the ADLs available emphasise is on visuals and analysis of software architectures as compared to their refinement and dynamism. It is worth pointing out that they are still growing. The integration of architectural based-tools used in the architecture description modelling and languages, standardisation of the notations, requirement elicitation, and evaluation methods are still considered as a major challenges facing SA description in general and ADLs development in particular.

2.6.2.2 Standardisation as a common problem in software architecture

There are common problems that impact the development of software architecture and its evaluation. This brings us to standardisation. Different and common architectural factors influence the development and evaluation of software architecture today concentrating on standardisations, which are:

- Commercial off-the-shelf products (COTS) versus the requirements satisfaction.
- Integration of different products versus environment challenges.
- Major companies buying out small companies that face integration and tool challenges.

Garlan [2000], proposed three solutions to the three challenging factors above. The solutions were as follows:

- Creation of industry standards (e.g. component based standards, UPDM).
- High-level architecture standards (e.g. IEEE-42010, DoDAF, MoDAF).
- Standardisation of notation and tools (e.g. UML, SysML, ADLs).

All his three solutions enforce standardisation in general for the architecture development process and evaluation while taking into account environmental challenges. For example, according to Babar et al. [2007] “Describing architectures is a big challenge. We use UML for this purpose, and UML tools evolved to have better integration with documentation packages like MS Word. Also, the survey reported by Malavolta et al. [2013], indicated the needs for better standardisation, integrability, and mature tools, in order to improve the description of SA and make its artefacts more useful in industry.
2.7 CONCLUSION AND CONJECTURE

Hence, tools that enable architects to design an architecture and then put the design into a format that is comprehensible to business people are important. Also, there is more need for stable integrated environment that incorporates modelling, text composition, and knowledge management features for supporting architectural practices, which could be improved by using standardised languages and better tools. For instance Artisan tool can generate all the architecture description and models into World format and the table into Excel format. However, the generated report format, needs more effort by the Artisan vendor (PTC Inc.), to be able to produce better reports.

2.6.2.3 Architecture modelling Challenges

The use of modelling approaches, such as MDA to develop software architecture and its evaluation has its challenges, France et al. [2007] explained the challenges that face MDE approaches (including MDD and MDA) during the Future of Software Engineering-(FOSE) workshop.

The categories below show the challenges that researchers have faced when trying to make the MDE vision a reality:

1. Modelling Language challenges; these are the problems that result as a failure to provide support for creation, the use of problem-level abstractions in modelling languages, and during the models analysis.
2. Separation of concerns challenges; the challenges here are as a result of problems associated with modelling systems that use multiple overlapping viewpoints to analyse heterogeneous languages. Thus, we should pay attention during this study, while proposing a solution, because we could face the overlapping problem between viewpoints.
3. Model manipulation and management challenge; this is as a result of problems linked with:
   1) defining, analysing, and usage of model transformations; 2) while trying to maintain traceability links between different model elements that support evolution and round trip engineering; 3) maintaining consistency among different viewpoints 4) tracking versions and lastly; 5) use of models during runtime.

More discussion about how MDE could influence the architecture evaluation has been illustrated in several resources such as model driven software development, Rech et al. [2009].

This suggests that the development of an architectural evaluation model/framework/profile, will be critically influenced by many factors that have to be considered during this research program.

2.7 Conclusion and Conjecture

In summary, this chapter draws attention to the challenges facing software architecture description and evaluation. The effort being put into this problem area, increases the importance of finding an optimal solution. That is still a long way off, but nonetheless reachable.
Currently the precision of SA documentation, description, evaluation, tools, frameworks, and models vary from language to another. However, there are no general standardised automated evaluation mechanisms available, which could deter the utilisation of SA artefacts. Information presented in this chapter regarding the current SA standardisation, integration, evaluation, and modifiability increases the necessity for more investigation and proposals for new solution concepts.

The main contribution, from this research, to addressing these issues is to i) target software developers’ minds and knowledge to improve their awareness and develop a common vision regarding new SA and SAE challenges; ii) offer new conceptual solutions, through the results of deeper analysis; iii) share additional knowledge gained from employing survey and field study methods; and iv) prove the problem involvement in industry through an analysis of a known reference architecture.

An approach concentrating on improving human (software developers) knowledge and offering a new initial SAE concept, based on interdisciplinary knowledge, which can be developed using several research methods, frameworks, and standards, is sought, in order to provide better SAE techniques to help in building better computerised products.
Part II

Contribution
The dilemma of Software Pattern descriptions with partial solution

It is commonly said that a pattern, however it is written, has four essential parts: a statement of the context where the pattern is useful, the problem that the pattern addresses, the forces that play in forming a solution, and the solution that resolves those forces.

Fowler [1997]
3.1 Introduction

The aim of the development of a Software Pattern (SP) is to provide a reliable and reusable framework for solving similar (software) problems within distinct contexts. To accomplish this objective competently, it is imperative to document patterns effectively, in order to facilitate the comprehension of their concepts to their users, thereby encouraging their utilisation over and over again. Thus, the documentation of SPs needs to be explicitly explained, together with their relationships with any Quality Attribute (QA) that they support or hinder, in order to better satisfy the implementation of stakeholders’ requirements.

This chapter illustrates how SPs are inconsistently defined, categorised, and linked to QAs, through deep analysis of six well-known SPs resources. Also highlighted are some important factors that impact pattern usability, followed by a proposed solution.

In order, to provide a reliable method for maintaining and easily representing the research work, I have created a database application containing all required information. This database should serve future research endeavours and thence help in developing software/systems with less confusion and predictable characteristics.

Is this study worthwhile?

The main SP issues in relation to Software Architecture (SA) were started with a few questions that are illustrated in Figures 3.1a and 3.1b.

Also, this study received a positive response as per the findings of a questionnaire. The result of a bipolar scale (Likert scale) for the statement: “Studying relationships between patterns and quality attributes based on the current reliable SP references, and creating a database to store these relationships on the basis of standardised quality attribute definitions, is valuable knowledge”, is significant, where $t = 6.34$, p-value < .01; hence, there is strong agreement with this statement, which makes this work worthwhile.

3.2 Rationale of the investigation approach

This investigation of SPs description and their relations in regard to QAs was necessary for three reasons:

i. SPs are important components to Software Architecture (SA), because developers often utilise SPs to create their own software/system architectures.
3.2. RATIONALE OF THE INVESTIGATION APPROACH

ii. To emphasise the problem concept.

iii. To increase the value of the proposed solution to (software) pattern users.

*The investigation was accomplished through a phased approach as illustrated in Figure 3.2.*

![Figure 3.1: Visualizing the Problem Area.](image)

Figure 3.1: Visualizing the Problem Area.

![Figure 3.2: The main SPs investigation phases towards producing the solution.](image)

Figure 3.2: The main SPs investigation phases towards producing the solution.

*Seven analysis steps have been carried out to satisfy the above phases as described in Table 3.1.*

109
### Table 3.1: Descriptions of the 7-Analysis Steps for the Targeted Resources.

<table>
<thead>
<tr>
<th>Process #</th>
<th>Investigation Steps</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pattern Resource Selection.</td>
<td>Identifying the most widely and reliable resources within the field of software patterns through extensive literature review, so becoming the targeted resources for this investigation study, (phase 2)</td>
</tr>
<tr>
<td>2</td>
<td>Pattern Categorisation Approach.</td>
<td>Study and compare all categorization approaches within the selected resources, (phase 1)</td>
</tr>
<tr>
<td>3</td>
<td>Pattern Descriptions.</td>
<td>Study and compare the description of patterns between targeted resources in the domain of quality attribute relationships. This step includes the investigation of every resource and the way they define and categorise quality attributes in their descriptions, (phases 1 and 3)</td>
</tr>
<tr>
<td>4</td>
<td>Quality Attribute Approach.</td>
<td>Identify and select one of the best-standardized practices in the field for defining and categorising the quality attributes through further literature review. Then use the selected approach for identifying the relationship between SPs and QAs. Also, the same approach is used for comparisons between different quality attributes categorization schema within the targeted resources, (phases 1 and 3)</td>
</tr>
<tr>
<td>5</td>
<td>Creation of the Relationship Matrices.</td>
<td>Based on (SP and QA) descriptions that have been reported by the selected resources; a (SPs-QAs) relationship matrix for each resource have been developed. Also, general common matrix that include all individual matrices are created for better usability, (phase 4)</td>
</tr>
<tr>
<td>6</td>
<td>Creation of the Quality Attributes Categorization Tables.</td>
<td>Based on the information collected from steps 1–5, create comparisons tables for the QAs classifications, between selected QAs approaches and others, within the targeted resources, (phase 4)</td>
</tr>
<tr>
<td>7</td>
<td>Conflicts and Issues.</td>
<td>Based on the investigation steps 1–6, identify any relationship conflicts and issues within the descriptions of patterns on targeted resources, (phases 1 and 3).</td>
</tr>
</tbody>
</table>

The software pattern sources included in this study are:

i. *Gamma et al.* [1995] – the Gang of Four (GoF) book,

ii. *Buschmann et al.* [1996] – Pattern-Oriented Software Architectures (POSA)-V1,


3.3 Investigation Analyses

Figure 3.3 illustrates the references included in this investigation.

The selection of these six pattern sources is based upon my preliminary research and respondents’ answers to a survey questionnaire distributed in 2012, as part of the investigation phase one. Almost half of the respondents identified GoF and POSA books as their reliable, popular, and well-known software pattern references. Bass et al. [1998] and Schumacher et al. [2006] are included in this study as they tackle architectural styles, the security domain, and are good examples of pattern languages. Despite the age of some of the resources, they are all still utilised today, confirming their ongoing value and suitability for inclusion in this research.

3.3.1 Patterns and Quality Attributes Refinement

To create or describe a pattern we should understand the concept of patterns and follow rules or constraints to document them in the right way. To assess patterns against QAs, we should do the same to the QA concept. The rest of this section lays out the problems that exist within the concept and rules of creating and documenting software architecture patterns that have a direct impact on their utilisation and evaluation. Also, this section presents justifications for building a (SPs-QAs) relationship database, and some of the challenges that have been faced during its construction.
3.3.1.1 Problems Discovered within the Current Pattern Definitions and Terminologies

Numerous pattern definitions are suggested for varying contexts. It might therefore seem difficult to define patterns in commonly acceptable terms. However, it seems sufficient to say that a pattern is essentially the solution to a problem within a particular domain, which can be applied to help resolve similar problems in different contexts within the same domain. The definition of ‘context’ has evolved over time. For the purpose of this study I deem Dey [2001] definition as the most appropriate and probably the most widely accepted.

Dey defines ‘context’ as “any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between (for example) a user and an application, including the user and the application”. Also, Alshaikh et al. [2008] and Alshaikh [2011] PhD thesis provide a recent and deeper analytical study of context, its use in general, and in the software domain in particular.

Gabriel found the definition of a pattern as described by the GoF is “a solution to a problem in a context,” is unacceptable, Coplien [2014]. He believed that it failed to illustrate the significance of the concept, and may have even caused misinterpretation amongst software professionals. Also he believed that many of the existing pattern definitions were indistinct and did not accurately express the implications patterns have. He therefore proposed a new definition, amending an early version by Alexander [1979]: “Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces, which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves”.

Others, such as Buschmann et al. [1996], Fowler [1997], and Riehle et al. [1996], and Coplien [2014], have their own definition of ‘pattern’.

Most of the definitions share common key points with a few variations. Some are more elaborate than others or include further important aspects such as forces. Defining the driver forces and constraints in a pattern as a solution is an important step during pattern resolution, Alexander [1979].

Having different terminologies and names in real life to explain the same thing, often due to differences in cultural factors or language, is acceptable. However, this is inappropriate in the context of SPs, as it leads to confusion. It is therefore considered as an absence of standardisation, which can cause major challenges, Garlan [2000].

The terminologies shown in the Figure 3.4 are being used within the current literature, Graça [2017] and Fairbanks [2018]. For example, the Architectural-Styles term is used by Bass et al. [1998], Fielding [2000], and Capilla et al. [2016]. While, Architectural-Pattern is termed by Buschmann et al. [1996] and Richards [2015]. Also, Alexander [1964], Fowler [1997], Bass et al. [2013], and Comyn-Wattiau et al. [2016] refer to Conceptual-Model; and
3.3. INVESTIGATION ANALYSES

lastly, Conceptual-Pattern is used by Riehle et al. [1996] and Grone [2006].

Thus, the terminology problem persists, while the context of developments changes as shown in Figure 3.4, suggesting different concepts. But are they?

The philosophy surrounding the conceptual or architectural ‘model, style, and pattern’ in the aforementioned terms attempts to convey a single idea through various explanations, all of which share the concept, components, restraints, and relationships that focus on a high abstraction level. However, the conceptual models should be explained further through detailed descriptions, in order to be able to move from an architectural context to a design context and so forth.

All the terminologies shown in Figure 3.4, do have the same concept of pattern, which is to provide a solution for a specific problem context.

Although, minor differences exist, which are based on each author’s visions and usage of patterns within various development contexts, (e.g. architecture level or design level).

However, common terminologies surrounding patterns and concrete descriptions might lead to improving the utilisation and understanding of SPs, which should minimise the confusion in the mindset of their users. Hence, these variations are challenging factors for this study.

The same discussion also applies to the design phase as briefly discussed below.

Figure 3.4: Terminologies of “Pattern” within software development lifecycle.
Alexander defines design as “a process of synthesis, a process of putting together things, a process of combination”, Alexander [1979].

According to Buschmann et al. [1996], “design patterns depict frequently occurring arrangements of interacting components, thus helping to resolve design dilemmas in a given frame of reference”. What this essentially suggests is that a pattern cannot be translated into code, but rather the pattern should be moulded in a way that it provides a solution to the problem.

Currently, software developers can select required patterns in the form of code, for example State Pattern, coded in Java, Barth et al. [2007, pp. 187-189]. Also, there are numerous patterns on the Internet, which are coded in different programming languages and are ready for use. In my opinion, I think that restricts the generalization concept of pattern description.

Therefore, I agree with Buschmann et al. [1996], that patterns should be highly generic with textual explanations in addition to block and connector diagrams, in order to support higher reusability in multiple contexts and better understanding. However, the textual explanations and the block and connector diagrams should not be arbitrary. Also, they should be applied within a common standardised procedure, description language and/or a framework. Consequently, their usability factor will increase due to the clarity of their descriptions. Also, divergences in pattern interpretation does not take them out of their context or main objectives.

Various definitions (rules) of design patterns that convey a diversity of terminology and description are apparent when comparing the definitions of Alpert et al. [1998], Wolfgang [1994], Coplien et al. [1995], Gamma et al. [1995], and Buschmann et al. [1996].

To conclude, the concept of ‘pattern’ can be used for describing an architecture, design, and implementation. What’s different then? It’s the diversity of the context. Hence, reducing pattern documentation conflicts, needs more research and standardised procedures, to help increase their effective use. Possible research includes research on the practices used to apply patterns, empirical research on industrial software projects concerning patterns usage, and empirical research that compares the latest documented SPs descriptions and if they applied differently in the real world.

3.3.1.2 Problems Discovered within Current Pattern Categorisations

One manifestation of SPs diversity is their categorisation. A categorisation outline is employed to organise the patterns as a collection so as to make them accessible for searching and storing by users.

The classification approaches for the investigated resources are:

• The first and the second volumes of the POSA patterns are based on two primary categories: pattern and problem categories. The pattern category is subdivided into 3 types within both volumes, while the problem category is organised into 10 types in POSA-V1, and 4 types in POSA-V2.
3.3. INVESTIGATION ANALYSES

- POSA-V3 patterns are based on 3 primary categories within the domain of typical resource management life cycles. These categories are resource acquisition, resource life cycle and resource release.

- POSA-V4 patterns are categorised on the basis of technical topics and distributed systems. Bear in mind, that POSA-V4 has been added into this section for the purpose of classification study only, in order to emphasise the existence of SP categorisation problem.

- The GoF team, however, used a different approach, classifying patterns based on purpose and scope. ‘Purpose’ is further sub-categorised into creational, structural, and behavioural, while ‘scope’ is divided into categories of classes and objects.

- The SEI book by Bass et al. [1998] contains architectural styles that are categorised on the basis of respective subjects and relations. Bass et al. [1998] describes thirteen different styles, of which the five primary styles are independent components, data flow, data-centre, virtual machine, and call and return. The primary styles signify the relationships amongst the sub-styles and their respective topics. Taking that into account, the Bass et al. [2013] version does not mention their styles’ categories.

- The book on security patterns by Schumacher et al. [2006] comprises pattern categories bearing reference to enterprise and system levels within the security domain, and its related to engineering and operations activities at all levels.

Based on this study, the context of technical topics in (POSA-V4) is the same as the context of technical problems, as well as the problem category, which are recognised within volumes 1 and 2. For example, the ‘From Mud To Structure’, is described as a problem category in POSA-V1, and as a technical topic in POSA-V4.

Also, another example of the categorisations confusion, is concerning the Interpreter pattern, where GoF considers this pattern as a design (behavioural) pattern, but the SEI group considers it an architectural (virtual machine) style. So, what is the Interpreter pattern, and does this affect the reusability of this pattern? Can we use the same pattern, explained by GoF in the context of a virtual machine as explained by SEI group, or do we need to adjust it to fit the new context?

By comparing the targeted resources mentioned above, it is clear that there is no common approach for categorising patterns. However, the ‘problem’ as a category concept, is shared between many pattern books, although under a variety of names, for example, it is named ‘purpose’ in GoF book; ‘problem’ in POSA-V1 and V2, ‘technical topics’ in POSA-V4, and as ‘main style or related subject’ in SEI work.

This lack of a common classification, particularly for technical topics, such as software patterns, can end up complicating things for users, researchers, and readers. Therefore, when users seek appropriate patterns for resolving certain real-life issues, they are confronted with
different guides and classifications for what are essentially the same patterns. Whilst, this can assist the users in employing the patterns in diverse contexts, it may also contribute towards making the reuse factor of patterns more complex, unmanageable, and less efficient.

3.3.2 The Variation Concept as a Problem within QAs

There are many different schools of thought regarding the management of QA and how they can be addressed effectively, such as ISO, SEI, U.S. Department of Defence (DoD), and IEEE) standards, Futrell et al. [2002]. Hence, there are challenges that arise when quality has to be defined in the real world. This section tries to demonstrate in brief the difficulties that arose during this study from the QA documentation variation viewpoint.

According to Mitra [2008] and references therein pertaining to Juran and Gryna (1993), the Crosby (1979), IEEE-1061, and ISO-9126 (including the superseded ISO-25010)\(^1\), each has their own individual concept of quality. Dr. Ronald Petrasch [1999] argues that there are variations in QA definitions that are acknowledged by both the community and researchers involved. The presence of different concepts of quality amongst different people and communities illustrates that there are variations within the definitions for each QA that are likely to be inconsistent.

Thus, small variations within QA definitions could increase the difficulties in defining and evaluating software architecture including its styles/patterns.

As with SPs, variations exist in QA categories and terminologies, depending on the domain in which they are applied. People have designed different ways to classify QAs using different approaches.

Evidence of such variations introduced to improve the software measurement domain include:

- Quality in use: ISO-25010 has 5 characteristics instead of 4 in ISO-9126. Satisfaction, efficiency, and usability were added to the latter, while compliance and productivity were deleted.
- Internal and external characteristics and sub-characteristics integration with two new characteristics: compatibility and security, Desharnais [2013]. Also, the comparative study of software quality models reported by Suman et al. [2014] demonstrates a similar example.

However, these aspects will not be discussed within this research. The focus in this study is to explore the differences in current QA and SP documentation and to demonstrate the issues associated with these differences concerning their relationships with each other. An elaborated example will explain this matter in Section 3.4.

\(^1\)Even though, (ISO-25010) was released after this statement was published, the earlier standard is included here, because the statement is applicable to both standards.
3.4 Conflict Example - (Proxy Pattern)

This example has been constructed to depict and illustrate the issues discussed in Sections (3.3.1 and 3.3.2). It is a comparison between the GoF and POSA-V1 Proxy pattern documentations. This comparison shows some differences that led to confusion and reduction in the utilisation of SPs, as demonstrated in Figures (3.5 and 3.6).

![Proxy pattern categorization schema, variants, and relationship with quality attributes By Gamma et al. (1995) – (GoF)](image)

Figure 3.5: The GoF approach for classifying and describing Proxy pattern, includes all variants and relationships with quality attributes.

The definition of the Proxy pattern has similarities in both resources. While POSA-V1 does elaborate further in their description, there are, nonetheless, other differences between both approaches such as:

1. Number of instances or variants.
2. Their primary and secondary categorisations.
3. The **Proxy Pattern** relationship with QAs.

The GoF divides **Proxy Pattern** into four (4) variants (Figure 3.5):

- Remote.
- Virtual.
- Protection.
- Smart Reference or (Pointer).

In contrast, the POSA group divides the **Proxy Pattern** into seven (7) variants namely (Figure 3.6):
Proxy pattern categorization schema, variants, and relationship with quality attributes By Buschmann et al. (1996) – (POSA – Volume 1)

<table>
<thead>
<tr>
<th>Pattern Design</th>
<th>Problem Access Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secondary Categorization</td>
<td>Primary Categorization</td>
</tr>
<tr>
<td>(POSA V1) “Proxy pattern variants”</td>
<td>Quality Attributes</td>
</tr>
<tr>
<td>Virtual</td>
<td>Lower Cost</td>
</tr>
<tr>
<td>Protection</td>
<td>Usability</td>
</tr>
<tr>
<td>Remote (Ambassador)</td>
<td>Security Performance</td>
</tr>
<tr>
<td>Synchronization</td>
<td>(apply to all variants)</td>
</tr>
<tr>
<td>Cache</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Firewall</td>
<td>(apply to all variants)</td>
</tr>
<tr>
<td>Counting</td>
<td></td>
</tr>
</tbody>
</table>

The common 'variants' (with yellow boxes in Figure 3.5 and Figure 3.6) between both classifications are **Remote, Virtual and Protection**. An important question here is, which QAs are supported or hindered by the variants in both references?

Figure 3.5 shows that all GoF Proxy pattern variants support ‘lowering cost’ as a QA, and Virtual and Protection variants support ‘optimization and security’ respectively.

The POSA team on the other hand considered all Proxy pattern variants, including common ones, to be supportive of the QAs usability, security, and performance, but hindered the QA ‘efficiency’.

The above divergence in categorisations schema, variants, and relationships between
3.5 Summary of the issues discovered by this study

The most important issues revealed by the above four sections (from 3.1–3.4) are listed with a few more examples in this section, as follows.

- There are no standardised definitions or categorisations of QAs that are presented by Gamma et al. [1995]. Their approach instead focuses on the explanation of how patterns can be used to support claimed QAs. They used their own (largely non-standard) words and examples to explain QAs in the context of software patterns.

- ISO-25010 standard and its predecessor ISO-9126, the POSA Books, and SEI Bass et al. [1998], define QAs somewhat differently using varying vocabularies, although the concepts of their definitions are largely similar for each QA. However, definitions vary in their sentence structuring, terminologies and how many features or constraints are included. I believe, that any additional features or constraints added to any QA definition should be considered as a prerequisite that needs to be fulfilled, to achieve that QA with all its characteristics. This could have an influence on this research evaluation framework concept in the future. As a result, the above variations in the QA descriptions could have an impact on the overall evaluation process for any system or structure (e.g. patterns), and may cause a conflict between development teams if they use non-common descriptions for the desired quality requirements.

- ISO-25010 standard and its predecessor ISO-9126, the POSA Books, and SEI Bass et al. [1998] present different QA categories. For example, ISO-25010, ISO-9126, and POSA Books, each have ‘Reliability’ as one of their main categories, but they differ in their sub-categories as illustrated in Figure 3.7. It is very likely then that we will experience differences when trying to satisfy or validate the ‘Reliability’, using both approaches.

- One of the biggest causes of confusion and difficulty in traceability is the use of different...
names for the same patterns or the one name for different patterns. For example, GoF team explain that *Adapter and Decorator* patterns as two different patterns, which they are. However, both have been identified as a *Wrapper* pattern. *It is neither logical nor user-friendly for the same pattern to have different names or different patterns to have the same name, making it hard to identify, trace, and apply.* It is understandable to have a variety of names if the pattern has individual instances or variants, such as the Proxy variants example discussed earlier. There are other examples of this “documentation problem” where the same pattern has various titles: *Publisher-Subscriber, Observer, and Dependent* are all different names for the same pattern.

*Indeed there are 8 different names described by Schumacher et al. [2006] for Check-Point pattern alone!* They are:

Policy Definition Point (PDP), Policy Enforcement Point (PEP), Access Verification, Holding off Hackers, Validation and Penalization, Make the Punishment fit the Crime, Validation Screen, and Pluggable Authentication.

However, GoF and POSA books have provided something as a solution to this problem, by introducing “Also Known As” section during their pattern descriptions. Other resources such as *Schmidt et al. [2001] and Bass et al. [1998]*, however do not acknowledge alternative names in their work.

- Some resources include the same patterns with the same names and definition, but with different QA relationships. For example, in POSA-V1, the Piping and Filtering pattern supports Testability and Exchangeability, whereas SEI book lists it as supporting Maintainability and Usability. Questions therefore arise as to which QAs the pattern truly supports, and how these different conclusions have been reached. Not forgetting that QA relationships seem arbitrary, and the answer most probably lies with the differing experience and observations of the pattern authors, or because there is still a lack of proper methodology to capturing and documenting patterns, which I believe (from experience), is the case.

Using expert knowledge regarding recurring problems to provide feasible solutions to the community relies on good standardised documentation, as recommended by *Garlan [2000]*. Standardisation helps decrease the challenges facing software development, preventing user confusion. Also, representing experts’ opinions regarding SPs-QAs relationships in one place with comparison to known QA standards should help minimise the confusion and increase developers’ understandability, as explained in the following section.

### 3.6 Proposed solution

A proposed solution is presented in this section, based on SP-QA investigation.
3.6. PROPOSED SOLUTION

The objective of this solution is to produce SPs and QAs definitions, categorisations, and relationships discrepancies between reliable resources in a new easy representation, in order to:

1. Round up all SPs-QAs required information from all selected references into one place providing summaries for numerous resources, to facilitate the process of comparing their descriptions, and to increase the utilisation and evaluation of SPs by showing the main conflicts.

2. Enthuse software developers and researchers to be able to contribute to the problem domain, in order to improve software/systems development process by improving SPs documentation.

3. Improve SA description and SAE domain by knowing SPs-QAs relationship discrepancies before utilising SPs.

To follow standardised documentation as recommended by Garlan [2000], I include the ISO-9126 quality model and the aforementioned resources in Section 3.2 that are the references from which to build the relationship matrices between both SPs and QAs.

Please note that, the quality standard ISO-25010 is not included, because this investigation was started and relationship matrices were developed before it was published.

However, the SPs-QAs relationships were built based on both individual QA assessment and a hierarchy schema. Thus, the proposed solution is still applicable if the ISO-25010 standard is used for individual QAs rather than the hierarchical relationship between a main QA category and its sub-categories.

For example, due to the sub-category differences between the two standards, the Reliability QA in ISO-25010 will not be fulfilled (in this study) by Availability, but it will be fulfilled by all the other three QAs as illustrated in Figure 3.8. Whereas, Reliability within ISO-9126, will be fulfilled by all its sub-categories.

Figure 3.8: Reliability sub-categories differences between ISO-9126 and ISO-25010.

The solution main features are:

- identification and categorisation of 168 patterns, and relationships between 120 (SPs) and 50 (QAs);
CHAPTER 3. THE DILEMMA OF Software Pattern DESCRIPTIONS WITH …

- creation of a general matrix for (SPs-QAs) relationships, individual matrices, and pattern categorisations tables;
- QAs description tables consisting of definitions, comments and their relationships with related SPs;
- SPs description tables consisting of definitions, alternative names, comments and their relationships with QAs;
- a contrast table of QA classifications between POSA, SEI and ISO-9126; and
- a sustainable representation for this study’s results provided through a database with highly usable navigation.

To avoid repetition within the following sub-sections, I will try to explain necessary information only, because most of the figures included are clear and self-explanatory.

To conclude, the following sub-sections describe the proposed solution through its tool (database) pages, which provide an excellent single point of reference for all particular users through its User Interface (UI).

3.6.1 Functionality description

The database application is a comprehensive solution for creating, editing and viewing all information collected and described in the previous three phases.

In an endeavour to provide a “Best-in-Class” way of maintaining and representing the research work, it was crucial to develop an auxiliary tool that would deliver good reporting service, and reduce creation and editing difficulties. There was also a need to gear-up to the next level, having a systematised and integrated monitoring interface, suitable for use in ongoing research activities pertaining to a similar field.

Some of the proposed solution repository features and benefits are:

- Usability Impact: Enhance user satisfaction with benefit of being highly convenient; information can be accessible over intranet & internet.
- Operational Impact: increase efficiency and effectiveness.
- Reusability Impact: can be customised for any other research activities in the future.
- Security features – auditing login/logout information, comprehensive user management and password policy.
- Can be used to host other research work results and data.
- No limitation on the number of users.

The overall database structure, matrices, tables, and graphical models are illustrated in Figure 3.9.

3.6.1.1 Pages descriptions

1. **Home**: The landing page for the application. It has 2 tabs as shown in Figure 3.10:
3.6. PROPOSED SOLUTION

Figure 3.9: Overall structure of matrix pages and tabs.

- Overview: describes the application in general.
- DB Schema: shows the graphical DB Schema of the research work.

Figure 3.10: Landing page, which describe the application and its pages functionalities.

2. **Pattern List**: landing page for the pattern list, which include the *general pattern list* that shows all the patterns available in alphabetical order as shown in Figure 3.11. The pattern list tabs include:
• SEI Patterns: displays pattern list for SEI matrix
• GOF Patterns: displays pattern list for GoF matrix
• POSA1 Patterns: displays pattern list for POSA1 matrix
• POSA2 Patterns: displays pattern list for POSA2 matrix
• POSA3 Patterns: displays pattern list for POSA3 matrix
• Security Patterns: Displays Pattern list for Security matrix

3. **Patterns Categorisation**: the categorization overview is the landing page, which shows the general categorisation menu diagram as illustrated in Figure 3.3, Section 3.2. Figure 3.12 shows the drop down list for all individual SPs categorisation tables included.

Furthermore Figure 3.13 shows the individual categorisation table for GoF, which includes:

- Name of the reference.
- Number of patterns included (24 patterns).
- Context of use, which briefly describes the scope of the reference.
- Categorisation of patterns based on the reference classification theme, obtained during this investigation. In this case it is based on primary and secondary categories.

> One look at the table in Figure 3.13 will show it is a summary of the GoF patterns categorisation, which supports user efficiency, effectiveness, and usability during pattern selection. Each pattern within the table is linked to its individual pattern description table for more information.

4. **Pattern description table**: each pattern table has several fields that explain pattern name, other names, context, references, relation types with related QAs, and a comment field to add any comment if required, as shown in Figure 3.14 for the CheckPoint pattern.
In the future, the patterns description tables should include all the information shown in Table 3.2, which is an example of summarising information for the Check Point pattern. The table gives an idea of what information may be needed to evaluate patterns. All the table fields are modified from the original form as reported by Reussner et al. [2005, p154], except for pattern name and type fields. Such a table could be improved and also evolve with time.

5. **QA description table**: each QA table has several fields that explain a named quality, definitions from related sources, references, relation types with related SPs, and a comment field to add notes if required, as shown in Figure 3.15.

6. **Patterns Vs Quality attributes**: *A glimpse of the general matrix page in Figure 3.16*
Also, the table shows the (SPs) related to Efficiency.

summarises all possible relationships between included SPs and QAs. Each pattern and quality within the matrix is linked to its individual description table for more information. However, individual matrices represent specific pattern and quality relationships as stated in a particular resource.

7. Edit existing relation: this page can be used to edit the relationships and Comments on the existing patterns/quality relationships, as illustrated in Figure 3.17.

8. Create new relation: create page can be used for the following:
3.6. PROPOSED SOLUTION

Table 3.2: Example of extracting the pattern data in a format suitable to assist evaluation, after modifications.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Pattern Type</th>
<th>Other Names</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Check Point Security Pattern</td>
<td>PDP, PEP, Access Verification, Holding off hackers, Validation and penalisation, Make the punishment fit the crime, Validation Screen, Pluggable Authentication.</td>
</tr>
<tr>
<td>Brief description</td>
<td>The Check Point Pattern provides effective I&amp;A to the system including a centralised and enforced security policy. It defines the interface to be supported by concrete implementation to provide the I&amp;A services to other security patterns such as Single Access point.</td>
<td></td>
</tr>
<tr>
<td>Context of use</td>
<td>Helps developers to design systems with an effective I&amp;A and access control mechanism easy to deploy and evolve.</td>
<td></td>
</tr>
<tr>
<td>Problem description</td>
<td>Secure the application from attacks, and provide appropriate actions when under attack.</td>
<td></td>
</tr>
<tr>
<td>Suggested solution</td>
<td>Encapsulate the security policy to be applied. Provide the application with an I&amp;A mechanism.</td>
<td></td>
</tr>
</tbody>
</table>
| Forces | 1. Providing a method to authenticate users and validation about what they can do is important.  
2. An appropriate message should be provided when users make mistakes.  
3. Too many consecutive mistakes at authentication by a user indicate an attack and should be dealt with.  
4. One place to refer to for authentication and authorisation to manage the security and reduce the system complexity.  
5. Response actions to mistakes should be dependant on the severity level.  
6. Use of small proven components for security, such as this pattern. This increases system maintainability, modifiability and reliability.  
7. Encapsulation of the security policy increases system independence and flexibility. |
| Available tactics | 1. Create Failure algorithm mechanism with a counter, for example implement security checks at the single access point (e.g. password).  
2. Include Error handling mechanism, to manage mistakes based on the severity of the error.  
3. Encapsulate the security algorithms in one component and make it configurable to be easy for maintainability and modifiability according to changes in requirements. |
| Affected Attributes | Positively: Flexibility, Maintainability, Modifiability, Reliability, Testability.  
Negatively: Simplicity in algorithms, interface, and configuration. |
| Supported general scenarios | S1: Warning message should be provided if the user makes non-severe mistake.  
S2: Force an abort of the logging process or quit the program at the level of high severity.  
Sn: … |

- Creation of new pattern.
- Creation of new quality.
- Creation of new relationship among patterns, qualities, and matrices with addition of specific comments; as shown in Figure 3.18

9. **Search Facility**: search page is used to perform the following functions:
CHAPTER 3. THE DILEMMA OF Software Pattern DESCRIPTIONS WITH …

Figure 3.16: Snapshot of general matrix that shows the relationships between SPs and QAs; where - $S \Rightarrow Support$, $H \Rightarrow Hinder$, and $B \Rightarrow Both$ (conflict).

Figure 3.17: Edit existing relation page.

- Search Pattern: at the click of the search pattern button, a description table regarding a chosen pattern will be displayed, including its relationships with different qualities, and specific comments.
- Search Quality: at the click of the search quality button a description table regarding the chosen quality will be displayed, including its relationships with different patterns, and specific comments.
- Search Conflict: at the click of the search conflict button, all the possible conflicting
3.6. PROPOSED SOLUTION

Figure 3.18: Create new relationships between SP and QA-(QA information page- step 2). Relationships between qualities and patterns will be displayed, as illustrated in Figure 3.19.

Figure 3.19: Search relation page, either by pattern or by quality attribute. Also, it could search all conflict relations within the database.

Taking into account, that I am familiar with both the patterns references (books) and the
database, I have conducted an iteration trial to find a SP-QA relationship from both the references and the database for comparison purpose, where the targeted SP-QA relationship is generated randomly by me. The result is that the time taken to find a relationship in one actual reference – (book) is more than double that to find the same relationship within the database.

To conclude this section, the proposed solution gathering of SP-QA relationships in one repository could help pattern users to save their time and to gain other benefits and services provided by the database.

### 3.7 Related work

Currently, most software pattern resources describe patterns based on authors’ experiences and observations. Some of these resources have pointed explicitly to the relationship between each pattern and its (apparent) QAs, i.e., *Gamma et al.* [1995] and *Buschmann et al.* [1996], while others do not; such as *Schmidt et al.* [2001] and *Fowler et al.* [2003].

However, there are a few sources that analysed the identification of the relationships between SPs and limited QAs, by using a scientific methodology (based on measurements and metrics), *Freitas* [2009], such as the work done by *Kim et al.* [2006] and *Zayaraz* [2010]. Both these approaches have been described in Chapter 2, and both methods provide a good step towards building a concrete framework for SPs-QAs relationships.

Also, there are different evaluation studies for patterns that concentrate on a specific aspect, such as the development of a set of assessment criteria for a design pattern for evaluation and comparison purposes as reported by *Khwaja et al.* [2013]; and the categorisation of empirical studies that focused on software pattern application done by *Riaz et al.* [2015]. In addition, there is a study that tackles the effectiveness of *AspectJ* and Java programming using design patterns to capture functional and non-functional elements, *Teebiga et al.* [2016]; and the formalisation and quantification guidelines approach for the *Strategy* design pattern developed by *Hummel et al.* [2017]. However, there aren’t any other studies reported yet, that match the study scope, objectives, process, and findings of this latter work.

### 3.8 Conclusion

This investigation of the relationship between QAs and SPs has highlighted two main issues.

First, the conflict variations within patterns practice and documentation in the current literature, which may be caused by different factors such as authors’ (in) experience and immaturity of the patterns in the field of software engineering. Second, there isn’t a formalised approach or process to be followed, for describing the relationship between patterns and QAs, or for categorising them in a more sensible, and formal/verifiable way. Both these issues have led to the

---

130

2 *AspectJ*: is an aspect-oriented programming (AOP) extension developed for Java.
existence of conflicting relationships between patterns and QAs, which decreases the utilisation of patterns by users.

There is no existing study that tackles these issues, investigates this problem domain with a clear process, and includes reliable references and standard quality models such as this research endeavour does. As a result, this work produces important comments through its findings, and a partial solution through its database repository, which assists users with searching, creating, deleting, or even modifying any SPs-QAs descriptions or relationships.

In future work, the QAs and SPs description tables will need further updating, improvement, and review for the current information, in order to enhance knowledge about patterns and QAs. Furthermore, they should also include forces, scenarios, and quality metrics, as well as other information deemed essential for comprehensive knowledge about software patterns and their QA relationships.

So, this study serves as an initial phase for the possibility of the development of an evaluation framework driven largely by metrics rather than individuals’ observations alone.

More research is needed and the next chapter reports of efforts to discover some of the factors that either close or widen the gap between software developers and the SPs domain.
Factors Influencing Utilisation of Software Patterns: A Questionnaire Analysis Result

A pattern is described as a solution to a class of problems in a general context. When a pattern is chosen and applied, the context of its application becomes very specific.

Bass et al. [2013, pp 247]
4.1 Executive summary

The variation in description of patterns within contemporary literature makes the explanation of SPs-QAs relationships complex and difficult to follow, as discussed earlier (Chapter 3). Doing nothing about this situation is likely to have the eventual result that developers will be deterred from utilising patterns and/or that they ignore quality attributes. Either of these scenarios may hinder the process of development significantly and/or the attainment of required system quality.

This study attempts to reveal and propose solutions for some of the current problems that are facing the effective utilisation of SPs during the process of software development. Unfortunately, there isn’t enough empirical research or evidence that is currently documented regarding this problem domain.

This chapter reports practitioners’ perceptions (with relevant experience) regarding different aspects of SPs, such as: 1) SPs usage trend through a survey method, in order to disclose reasons that motivate and/or demotivate their utilisation; and 2) the process and nature of their current documentation.

Consequently, further research is recommended to develop a framework/method and tools to support SPs documentation and usage, based on the significant results obtained by this study.

4.2 Introduction

Various studies have attempted to detect, examine, assess, and describe the relationship between styles/patterns and QAs. Every study takes a unique focus with a scope within a particular QA and field, such as Zayaraz [2010], Jung et al. [2006], and Zhu et al. [2004]. Nevertheless, hardly any such studies have pointed out the relationship between architecture (including patterns) and quality attributes within a broad scheme that could be used for the majority of QAs.

Thus, there is a rising need for documenting software architecture by researchers (and expert practitioners) in order to provide more guidance on the methods used to design architectures. Despite the recognition of this need, and the great efforts by Clements [2003], Bachmann et al. [2005], and Bass et al. [2013] or ISO-42010 framework, there is still research and work to be done to produce significantly better solutions for this problem context.

To avoid repetition and having laid down the background literature on this subject within Chapters (2 and 3), this chapter and its associated Appendix D tries to address the aforementioned problems and to confirm some of Chapter 3 findings.

A survey was conducted to identify the factors that are likely to aid (as opposed to hinder) developers with the effective use of patterns during their work. Thus the following objectives were paramount:
4.3 RESEARCH METHODOLOGY AND SURVEY PROCESS

- To establish the current trend for the utilisation of SPs in software development with the aim of defining a clear guideline and framework to facilitate the utilisation of these patterns in future.
- To determine the factors that facilitate or prevent developers from utilising SPs within software development.
- To help developers by proposing solutions for establishing the best practices for documenting SPs, with the additional aim of laying a foundation for further research.

4.3 Research methodology and Survey process

The following subsections are an overview of the survey methodology, process, analysis techniques, and reporting format, which are based on the guidelines and recommendations provided by Tang et al. [2006], Denscombe [2014], Linaker et al. [2015], and Mello et al. [2016]. They also apply to the surveys presented in Chapter 5.

4.3.1 Research technique and process

A questionnaire method was chosen as a data collection instrument for this research, due to its suitability in collecting feedback from a wide audience within a short time, Kitchenham et al. [2001]. Alternative methods for the research such as personal interviews would have consumed more time, while managing to reach out to only a small portion of the potential audience.

The research followed the processes illustrated in Figure 4.1, with the focus on uncovering how important it is for developers to use SPs with more efficiency and with QAs in mind. The questionnaire development, formatting, wording, number of questions, and time of compilation by respondents went through a necessary review process by several researchers, and recommendations given by the statistical unit at the ANU. A high level of confidentiality was exercised in treatment of the responses to the questionnaire.

![Figure 4.1: The survey process that applied for Chapters 4 and 5, after Kasunic [2005].](image)

4.3.2 Instrument questions

The survey is composed of twenty questions, which are divided into three main sections as illustrated in Figure 4.2. Each section has its own objectives as illustrated in Table 4.1.

Table 4.1: Summary of the questions and the objectives of each section
Section 1 (Q1–5)
Objective
Aims to form a relevant demographic of the participants, such as (field of expertise, years of experience, work sector, their familiarity with the survey domain).

Section 2 (Q6–9)
Objective
Reveal the factors that influence or affect the usage of SPs within the field of software development. Understanding these factors will shed light on the improvements and changes that need to be made in order to meet software developers requirements, regarding the use of SPs.

Section 3 (Q10–20)
Objective
Targeting the documentation of SPs, which could be one of the major reasons that could influence their utilisation within the process of software development process. In order to achieve the objective of this study, it will be necessary to determine the appropriate documentation practices that software developers would prefer. Thus, its valuable to explore the respondents’ opinion regarding their current documentation trends. Furthermore, this section includes four statements that propose some solutions to the problem domain.

The survey questions for the three main section are shown in Tables (4.2, 4.3, 4.4), respectively.

Table 4.2: Section 1. Background Questions (Questions marked with (*) are mandatory).

<table>
<thead>
<tr>
<th>Section number</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Q1–5)</td>
<td>Aims to form a relevant demographic of the participants, such as (field of expertise, years of experience, work sector, their familiarity with the survey domain).</td>
</tr>
<tr>
<td>2 (Q6–9)</td>
<td>Reveal the factors that influence or affect the usage of SPs within the field of software development. Understanding these factors will shed light on the improvements and changes that need to be made in order to meet software developers requirements, regarding the use of SPs.</td>
</tr>
<tr>
<td>3 (Q10–20)</td>
<td>Targeting the documentation of SPs, which could be one of the major reasons that could influence their utilisation within the process of software development process. In order to achieve the objective of this study, it will be necessary to determine the appropriate documentation practices that software developers would prefer. Thus, its valuable to explore the respondents’ opinion regarding their current documentation trends. Furthermore, this section includes four statements that propose some solutions to the problem domain.</td>
</tr>
</tbody>
</table>

Table 4.2: Section 1. Background Questions (Questions marked with (*) are mandatory).
### 4.3. RESEARCH METHODOLOGY AND SURVEY PROCESS

<table>
<thead>
<tr>
<th>Q3*</th>
<th>How many years experience do you have in total in the software development field?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5–10 (years)</td>
</tr>
<tr>
<td></td>
<td>10–15 (years)</td>
</tr>
<tr>
<td></td>
<td>15–20 (years)</td>
</tr>
<tr>
<td></td>
<td>20–25 (years)</td>
</tr>
<tr>
<td></td>
<td>Over 25 (years)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q4</th>
<th>Are you aware of software styles / patterns?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q5</th>
<th>How often do you use software styles / patterns during your work?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Never</td>
</tr>
<tr>
<td></td>
<td>Infrequently (&lt;10%)</td>
</tr>
<tr>
<td></td>
<td>Reasonably frequently (&gt;15% and &lt;50%)</td>
</tr>
<tr>
<td></td>
<td>Regularly (&gt;50% and &lt;80%)</td>
</tr>
<tr>
<td></td>
<td>Nearly always (&gt;90%)</td>
</tr>
</tbody>
</table>

Table 4.3: Section 2. Pattern utilisation questions.

- In your opinion what are the main factors that discourage the utilization of software patterns by developers? You may select up to two options.
  - No or few available references
  - Poor documentation of existing software patterns
  - Very little teaching of patterns in academic institutions or industry
  - No proof of the solutions provided by patterns
  - Unknown quality attributes for combining software patterns
  - Developing new solutions saves more time than searching for, and implementing the right patterns
  - Hard to integrate with other components or existing systems within the solution domain
  - Other - please specify: [ ]

- What are the main factors that encourage the utilization of software patterns by developers? Choose up to two options.
  - Easy to find the right patterns that solve the problems encountered
  - Most available references are clear and well documented
  - Easy to implement
  - Clear identification of quality attributes possessed by patterns
  - Other - please specify: [ ]
During your selection of patterns did you care about or consider quality attributes? Please provide comment, based on your answer.

**Q8**
- Yes – Why?
- No – Why?

Comments:

If you did used software patterns in the past, will you keep using them in the future? Please provide comment, based on your answer.

**Q9**
- Yes – Why?
- No – Why?

Comments:

---

Table 4.4: Section 3. Documentation of Software Patterns.

<table>
<thead>
<tr>
<th>Q10</th>
<th>What are your main software pattern references? Please provide two:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Reference one</td>
</tr>
<tr>
<td></td>
<td>2. Reference two</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q11</th>
<th>Do those references describe standardized process to create and explain patterns? If you chose “Yes”, please explain in the comment box.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes, please explain?</td>
</tr>
<tr>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>

Comments:

<table>
<thead>
<tr>
<th>Q12</th>
<th>Regarding the references you mentioned at 1 (above), have the relationships between the patterns and the quality attributes been identified and stated clearly?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Not sure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q13</th>
<th>Has the relationship between the patterns and the quality attributes in those references been proved scientifically or otherwise?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Not sure</td>
</tr>
</tbody>
</table>
### Q14
Have you identified any conflicting views among any references regarding some patterns and/or regarding quality attributes? (e.g., One reference might say, Proxy pattern supports performance, but another might say that the same pattern hinders performance). If you chose "Yes", Please provide, a Pattern name and references names
- Yes – please provide pattern name and references names.
- No

Comments: 

### Q15
Do you support having different names for similar patterns?
- Yes – please explain?
- No – please explain?

Comments: 

### Q16
Do you support standard documentation practices for software patterns?
- Yes
- No
- Perhaps
- Not sure

Please indicate your level of agreement with respect to the following statements:

### Q17
Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### Q18
Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree
Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge and should be freely available.

Q19

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.

Q20

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

4.3.3 Invitation mechanism and Instrument distribution

The target group for the invitation letters was professional software engineers, and as a consequence the letters were sent through expert referrals and personal contacts.

Participants could access the survey online, making the respondent process more efficient, easy to participate, and less expensive.

The survey was distributed to seven organisations, from academia, industry, and government, within six different countries, and all participated. The respondents geographical distribution is shown in Table 4.5.

Table 4.5: Geographical distribution of the survey respondents by country.

<table>
<thead>
<tr>
<th>Country</th>
<th>Number of respondents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>36</td>
</tr>
<tr>
<td>KSA (Saudi Arabia)</td>
<td>11</td>
</tr>
<tr>
<td>USA</td>
<td>2</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>1</td>
</tr>
<tr>
<td>China</td>
<td>1</td>
</tr>
<tr>
<td>Greece</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>52</td>
</tr>
</tbody>
</table>

4.3.4 Target population and Sampling technique

The inclusion criteria for participants were a software developer/engineer with typical work experience of five years or more, mostly in architecture, designing and programming.

In both surveys reported in (Chapters 4 and 5), the random sampling method was applied preliminarily by sending an invitation to different IT organisations or departments to fill out the questionnaire.
There were too few responses to analyse, partially due to time constraints, and also because less people were likely to respond to unfamiliar sources. Hence, I used other sampling technique options ‘Availability and Snowballing’, which are non-probabilistic methods. The Availability method operates by inviting available people, who meet the participation criteria, to respond to the questionnaire through familiar contact; then asking them to forward or nominate others to participate in the study, i.e., the Snowballing method. One disadvantage of non-probabilistic sampling techniques is that the target population generality is limited. Thus, the results cannot statistically reflect the general population, Kitchenham et al. [2001] and Tang et al. [2006]. However, due to this survey’s investigatory nature and the selected targeted (software developers), I consider the selected sampling techniques were (at least) rational and given that similar techniques are frequently used in software engineering studies, applicable.

The sampling technique employed being non-probabilistic, the actual target population size was unknown and hence the sampling frame was unspecified. Therefore, the conventional sample size calculation formulas for random sampling were unusable, including the formula discussed by Cochran [1977], which is applicable for an infinite population using random sampling. Employing non-probability sampling does not depend upon a statistical computation of the sample size, instead good judgement is relied on to understand what is passable in particular situations, Denscombe [2014]. “In practice, the complexity of the competing factors of resources and accuracy means that the decision on sample size tends to be based on experience and good judgement rather than relying on a strict mathematical formula”, Hoinville et al. [1985, pp 73]. Also, the sample sizes (of 52 and 50 for surveys reported in Chapter 4 and 5 respectively) well exceed the limit of 30 samples in order to use the central limit theorem, Wayne [2016]. Based on the opinions from Denscombe and Hoinville et al, the target population and the data collection method used, I think the sample sizes are sufficient to meet the objectives of the two surveys.

### 4.3.5 Procedure of the analysis

Different analysis methods were used when applicable, such as descriptive statistics, t-test, and Analysis of variance (ANOVA). Also, summarisation tables and figures were developed for better visualisation. The Statistical Package for the Social Sciences (SPSS) tool was used during the analysis. This has been clarified in order to facilitate the notations of tables and figures.

In order to fully understand the results gathered from respondents, it was of paramount importance to perform a **Dimensional Approach**, which means not only analysis of the responses to each question, but also analysis of the dependency or relationship between several questions. The following phases have been carried out to analyse the questionnaire:

1. **Individual Analysis** *(One dimensional)*:
   
   Each question was analysed and the results discussed separately. Explanation for any significant result was described. Graphs and tables were created when needed.

2. **Matrices Analysis** *(Analysis of two or more questions together)*:
   
   Based on the goal and objective of this study and in order to find any significant correlations or de-
pendency between questions, analyses between questions were performed and analysed according to the matrix levels as follows:

- Two dimensional matrices analysis presented in Table 4.6:

Table 4.6: Two dimensional matrices analysis (only two questions will be analysed together).

<table>
<thead>
<tr>
<th>Quest.</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$X_2$</td>
<td>$X_3$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$X_3$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$X_6$</td>
<td>$X_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td>$X_9$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td>$X_{10}$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>$X_{12}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td>$X_{13}$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td>$X_{15}$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td>$X_{16}$</td>
<td></td>
</tr>
</tbody>
</table>

$n$ in $X_n$ stands for matrix number.
Example: For Question (1), two matrices can be written down as:
- Matrix 1: Q1 and Q5;
- Matrix 2: Q1 and each question from Q17 to Q20.
- Each matrix has a unique colour.

- Three dimensional matrices analysis shown in Table 4.7:

Table 4.7: Three dimensional matrices analysis (only three questions will be analysed together).

<table>
<thead>
<tr>
<th>Quest.</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X_1$</td>
<td>$X_2$</td>
<td>$X_3$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td>$X_{1,2,3}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$X_4$</td>
<td>$X_5$</td>
<td>$X_7$</td>
<td>$X_8$</td>
<td>$X_9$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$X_{6-9}$</td>
<td>$X_7$</td>
<td>$X_8$</td>
<td>$X_9$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td></td>
</tr>
</tbody>
</table>

$n$ in $X_n$ stands for matrix number.
Example: For Question (1), three matrices can be written down as:
- Matrix 1: Q1 and Q2 with each question from Q17 to Q20;
- Matrix 2: Q1 and Q3 with each question from Q17 to Q20;
- Matrix 3: Q1 and Q5 with each question from Q17 to Q20;

- Four dimensional matrices analysis illustrated in Table 4.8:

Table 4.8: Four dimensional matrices analysis (only three questions will be analysed together).

<table>
<thead>
<tr>
<th>Quest.</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
<td>$X_1$</td>
</tr>
<tr>
<td>2</td>
<td>$X_2$</td>
<td>$X_2$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
<td>$X_3$</td>
</tr>
<tr>
<td>3</td>
<td>$X_{3,4}$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
<td>$X_4$</td>
</tr>
<tr>
<td>4</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
<td>$X_5$</td>
</tr>
<tr>
<td>5</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
<td>$X_6$</td>
</tr>
</tbody>
</table>

$n$ in $X_n$ stands for matrix number.
Example: For Question (1), one matrix can be written down as:
- Matrix 1: Q1 and Q2 and Q3 with each question from Q17 to Q20.

Moreover, a database was created to organise, sort, explore, and do some primitive analysis regarding respondents’ answers, which could help the overall analysis process. These types of analysis couldn’t be obtained by SPSS tool, such as (Internet Protocol (IP) addresses, sorting the number of respondents by countries, discover and eliminate duplicated IPs addresses, sorting the answers to the questions by countries, etc.). Section D.4 shows some snapshots of the database.
4.3.6 Rationale of the selected analysis methods

Several statistical methods have been used to analyse the questionnaire responses. This section justifies the selection of the methods that have been used. Also, a professional statistician confirmed the suitability of the analysis methods that have been employed in the analysis.

4.3.6.1 Single Dimensional Analysis Methods

Various methods of analysis were used for different types of question with the aim of achieving the best and most accurate results. The first type of question was that with multiple choices allowing selection of more than one choice. These included Q1, Q6 and Q7. The rationale behind the analysis was to determine the frequency of choices and determine which were the most popular, based on the number of participants that selected the category. Responses were coded as 1 (selected) and 0 (not selected). A frequency table was constructed for the total count of 1 (selected) in each category and a bar chart was used to visualise the figures for each of the multiple response variables.

The second type of question was that with categorical responses without numerical meanings. These included Q2, Q3 and Q5. For this type of question descriptive statistics were not appropriate to provide an adequate analysis result. Frequency tables, bar charts, and pie charts were used in analysing these kinds of responses. Since the responses were mutually exclusive and so totalled to 100%, a pie chart was used to represent the results. Additionally, in order to detect any general tendency of respondents towards a specific response, a non-parametric Chi-square test was also performed on the results.

The third type of question includes questions with dichotomous responses (Yes/No). These included (Q4, Q8, Q9, and Q11-Q16). Just like the previous category, the responses to these questions were grouped into a frequency table and presented using pie charts and bar charts.

Finally, responses to Likert-type questions, Q17 to Q20, were analysed using descriptive statistics after encoding them into an ordinal scale of 1 to 5 where the extreme left of the scale was reserved for the opinion “Strongly Disagree”, extreme right of the scale was reserved for the opinion “Strongly Agree”, and the middle of the scale was reserved for the neutral opinion. A right-tailed one sample t-test was performed on each of these questions to figure out the trend of the participants responses.

4.3.6.2 Two-Dimensional Analysis Methods

To determine the relationship between responses to various pairs of questions, the methods selected for this analysis were based on the types of the individual questions forming each pair. The methods used for analysis and presentation of such results are described in full in Table D.1.

4.3.6.3 Three and Four-Dimensional Analysis Methods

Most of the variables considered in this study are of nominal scale, except for responses to the last four questions (Q17-Q20), which have an ordinal scale. Cross tabulation was deemed most appropriate for the nominal scale questions. In this case it was appropriate to use Chi-Square tests to determine the level of interdependence between any two or more variables. Responses to Q17 to Q20, which are on the ordinal scale, can be analysed using either t-tests for two variables or Analysis of Variance (ANOVA) for more than two groups. In this case a substantial variation is required to draw meaningful results. A variation could not have been achieved in this study, due to the limitation of 52 observations, with missing responses after Q8. As a result, the Chi-Square tests with this small number of observations using three and four dimensional analysis proved invalid. ANOVA tests produced non-significant results too.
this reason, it was concluded that three and four dimensional analyses were invalid, inappropriate, and non-significant. The Chi-square test results for cross tabulations under three dimensional analyses are all invalid due to low cell expected frequencies. ‘The Conservative Rule’ by Lewontin and Felsenstein in (1965), stated that no expected frequency should be less than 1. Also Cochran 1954, reported that “no more than 20% of the expected frequencies should be less than 5”, Everitt 1977. Even though most of the t-tests and F-test results are all valid in a statistical sense, however there were no significant results.

4.4 Findings and Recommendations

This section reports the findings of the survey based on the analysis of questionnaire responses. Discussions and important recommendations have been included for each finding in an effort to accomplish the research’s main aim and objectives.

Some of the results from the analysis of responses show significance as far as the questionnaire aim is concerned. Further, meaningful trends and conclusions were also drawn from some of the results, which may not been very significant in aiding to derive the best recommendations for future efforts to improve the utilisation of SPs.

With the study’s main objective in mind, the first step was to ensure that the basic requirements concerning the skills/experience of the respondents were met. Thus the preliminary task of this analysis was to establish the composition of the respondents in terms of their experience in the software development field, the sectors in which they had expertise regarding software development, their awareness of SPs, and their frequency of pattern usage. The credibility of the responses collected from subsequent questions would rely heavily on the composition of the respondents in terms of their awareness of SPs, the longevity of their expertise, and their respective work sectors.

Demographic data is illustrated in Table 4.9, which summarise the results of the (5) background questions.

Findings show that the largest percentage (67.3%) of the respondents to the questionnaire had their software development expertise in the field of coding. This percentage is based on the number of participants who responded to the question (N=52). The results show a broad distribution of respondents across all the fields of software development, which make the research comprehensive, hence reliable as they represented the opinions from most software development fields. The participation in this question was 100% because the question was compulsory for all respondents, with each one of them allowed to pick more than one option.

Closely related to the field of expertise, is representation across the main sectors of the economy with reference to software development. The objective was to determine any trend in the current utilisation of SPs within each sector. The findings show that the highest representation emanated from the academic sector, followed by Industry and Government respectively. However, a more extensive analysis using the Chi-Square test reveals that the difference between the distributions is non-significant at a 5% level. This leads to the conclusion that all the sectors are almost equally represented in this survey. The implication of this finding is that the challenges identified by this study regarding the utilisation and documentation of SPs in software development are applicable to all sectors, and the recommended solutions can be adopted by all of them.

For reliability of the results, it was necessary that the participants should have sufficient experience within the software development field (of no less than 5 years). The variation of the experience is sig-
4.4. FINDINGS AND RECOMMENDATIONS

Table 4.9: Frequency distribution of the questions regarding personal expertise and Chi-square test result for equality of group frequencies.

<table>
<thead>
<tr>
<th>Background questions</th>
<th>Frequency (%)</th>
<th>Chi-square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General field of expertise (N=52)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements elicitation / modelling / analysis</td>
<td>25 (48.1)</td>
<td>6.686 (0.3509)</td>
</tr>
<tr>
<td>Project management</td>
<td>25 (48.1)</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>20 (38.5)</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>29 (55.8)</td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>35 (67.3)</td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td>23 (44.2)</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>12 (23.1)</td>
<td></td>
</tr>
<tr>
<td><strong>Work sector experience (N=52)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Academia</td>
<td>21 (40.4)</td>
<td>1.885 (.3897)</td>
</tr>
<tr>
<td>Industry</td>
<td>18 (34.6)</td>
<td></td>
</tr>
<tr>
<td>Government</td>
<td>13 (25.0)</td>
<td></td>
</tr>
<tr>
<td><strong>Developers’ experience in software development field (N=52)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 – 10 (years)</td>
<td>32 (61.5)</td>
<td>19.077 (.0001)</td>
</tr>
<tr>
<td>10 – 15 years</td>
<td>12 (23.1)</td>
<td></td>
</tr>
<tr>
<td>Over 15 years</td>
<td>08 (15.4)</td>
<td></td>
</tr>
<tr>
<td><strong>Developers’ awareness of software styles/patterns (N=50)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>09 (18.0)</td>
<td>20.480 (.0000)</td>
</tr>
<tr>
<td>Yes</td>
<td>41 (82.0)</td>
<td></td>
</tr>
<tr>
<td><strong>Software style/pattern utilisation amongst software developers (N=48)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Never</td>
<td>11 (22.9)</td>
<td>8.833 (0.0316)</td>
</tr>
<tr>
<td>Infrequently</td>
<td>16 (33.3)</td>
<td></td>
</tr>
<tr>
<td>Reasonably frequently</td>
<td>17 (35.4)</td>
<td></td>
</tr>
<tr>
<td>Regularly or nearly always</td>
<td>04 (8.4)</td>
<td></td>
</tr>
</tbody>
</table>

significant at the 1% level as analysed using the Chi-Square test. This finding is still considered acceptable since developers with an experience as short as 5 years are considered experienced and likely been exposed to SPs, unless there are other factors that may contribute to their lack of exposure to SPs in their work.

Having met the first criteria for the selection of participants, the next task was to establish the participants’ awareness of and experience with SPs. Given that a majority of the respondents had at least 5 years of experience in the field of software development, they should be aware of SPs. This was confirmed by 82% of the respondents admitting to being aware of software styles/patterns. Thus, the survey overall results are more reliable. Notwithstanding, it is still surprising that 18% of the respondents were not aware of SPs despite their years of experience in the field of software development. This raised a question about the reasons behind their lack of knowledge regarding SPs, which could be related to the effectiveness of software styles and patterns documentation, teaching, or other factors, which I investigate further in Q6 and Q7.

Further than mere awareness of SPs, an unexpected result is that about (23%) have never used SPs. A Chi-Square test showed that the difference in the results was significant at the 1% level. This is a revelation and may indicate that there must be factors that either encourage or hinder the utilisation of SPs.
4.4.1 Discussion of Findings

This section interprets and discusses the significant and important results for both (one and two) dimensional analyses, based on the objectives listed in Section 4.2.

4.4.1.1 Individual Analysis

Having taken into consideration all the controls listed in Table 4.9, this section discusses important aspects of the analysis in a single dimension as reported in Table 4.10.

According to 45.7% of respondents, one of the major factors that hinder the utilisation of SPs within software development is: “very little teaching of SPs in academic institutions”. In addition to this, 41% believe that it is “hard to integrate patterns with other existing components in systems”. The first factor perhaps explains why nearly 20% of the respondents were not aware of SPs. Also, SPs poor documentation added a difficulty when searching for the right pattern to a solution, as opposed to developing one from scratch. Most of the factors hindering the utilisation of patterns support the findings discussed and effort presented in Chapter 3.

Based on the factors identified in Table 4.10, it must be recommended that more attention be paid to the incorporation of SPs and styles into the curriculum of learning institutions. This will automatically lead to proper documentation of the patterns in literature pieces, thus solving the third factor of lack of awareness of SPs. Once the teaching of software patterns is incorporated into curricula and documentation is done properly within a standardised framework for describing SPs, it is possible that most of the developers will embrace their utilisation of SPs and eventually discover that using SPs makes the process of software development much easier.

A second recommendation is to emphasise and focus on the factors that encourage the utilisation of SPs. Among these factors, the major one identified by the respondents is the ease of implementation of these patterns. However, the “ease of implementation” does not mean the “ease of integration” as mentioned earlier as one of the hindering factors in Chapter 3. The former is an encouraging factor, and the latter is a hindering factor. However, the lack of pattern interface mechanisms within the current literature and proper exposure and knowledge of SPs, could cause the reluctance by developers to embrace SP solutions, preferring instead to develop new solutions from scratch.

Another factor encouraging the utilisation of SPs is the clear identification of QAs that are possessed by SPs, as identified by 35.1% of the respondents. Currently, most of the relationships between patterns and QAs are identified by observation, as opposed to using a scientific methodology and proof, as discussed in the Chapter 3, Mei et al. [2017].

Similarly, 35.1% of the respondents believe that it is easy to find the right SPs to solve an (encountered) problem. Nevertheless, there are some challenges which are related to pattern documentation problems, such as duplicate names for different patterns. This has been highlighted in Chapter 3 as well. Finally, the respondents also acknowledged that most of the available references are clear and well documented, a factor I partially agreed with, although it was identified by the least percentage of the developers. However, SPs documentation needs more effort especially for identifying the relationships between them and QAs. Also, a possible reason for this could be the shortage in publicity of the available documentation. It is thus clear that in order to increase the utilisation of SPs, proper documentation, including identifying the relationships between SPs and QAs, is a key solution.

In an effort to determine how many of the respondents cared about taking, or actually took, into
### 4.4. FINDINGS AND RECOMMENDATIONS

Table 4.10: Frequency distribution summary for (Q6 to Q16).

<table>
<thead>
<tr>
<th>Questions (6-16)</th>
<th>Frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main factors that discourage the utilisation of SPs (N=46)</strong></td>
<td></td>
</tr>
<tr>
<td>No or few available references.</td>
<td>3 (6.5)</td>
</tr>
<tr>
<td>Poor documentation of existing SPs.</td>
<td>10 (21.7)</td>
</tr>
<tr>
<td>Very little teaching of patterns in academic institutions.</td>
<td>21 (45.7)</td>
</tr>
<tr>
<td>No proof of the solutions provided by patterns.</td>
<td>6 (13.0)</td>
</tr>
<tr>
<td>Unknown QAs for combining SPs.</td>
<td>9 (19.6)</td>
</tr>
<tr>
<td>Developing new solutions saves more time than searching for the right pattern.</td>
<td>10 (21.7)</td>
</tr>
<tr>
<td>Hard to integrate with other components or existing systems.</td>
<td>19 (41.3)</td>
</tr>
<tr>
<td><strong>Main factors that encourage the utilisation of SPs (N=37)</strong></td>
<td></td>
</tr>
<tr>
<td>Easy to find the right patterns that solve the problems encountered.</td>
<td>13 (35.1)</td>
</tr>
<tr>
<td>Most available references are clear and well documented.</td>
<td>12 (32.4)</td>
</tr>
<tr>
<td>Easy to implement.</td>
<td>15 (40.5)</td>
</tr>
<tr>
<td>Clear identification of QAs possessed by patterns.</td>
<td>13 (35.1)</td>
</tr>
<tr>
<td><strong>Developers who care about or consider QAs. (N=36)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>14 (38.9)</td>
</tr>
<tr>
<td>Yes</td>
<td>22 (61.1)</td>
</tr>
<tr>
<td><strong>Usage of patterns in the future by software developers. (N=37)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>10 (27.0)</td>
</tr>
<tr>
<td>Yes</td>
<td>27 (73.0)</td>
</tr>
<tr>
<td><strong>Developers’ opinions on whether those references describe standardized</strong></td>
<td></td>
</tr>
<tr>
<td><strong>process to create and explain patterns. (N=29)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>15 (27.0)</td>
</tr>
<tr>
<td>Yes</td>
<td>14 (73.0)</td>
</tr>
<tr>
<td><strong>Have the relationships between the patterns and the QAs been identified</strong></td>
<td></td>
</tr>
<tr>
<td><strong>and stated clearly within identified references? (N=29)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>5 (17.2)</td>
</tr>
<tr>
<td>Yes</td>
<td>7 (24.1)</td>
</tr>
<tr>
<td>Not sure</td>
<td>17 (58.6)</td>
</tr>
<tr>
<td><strong>The relationship between the patterns and the QAs in those references</strong></td>
<td></td>
</tr>
<tr>
<td><strong>been proved scientifically or otherwise. (N=28)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>7 (25.0)</td>
</tr>
<tr>
<td>Yes</td>
<td>5 (17.9)</td>
</tr>
<tr>
<td>Not sure</td>
<td>16 (57.1)</td>
</tr>
<tr>
<td><strong>Developers who had identified conflicting views among any references</strong></td>
<td></td>
</tr>
<tr>
<td><strong>regarding some patterns and/or regarding QAs. (N=25)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>24 (88.9)</td>
</tr>
<tr>
<td>Yes</td>
<td>3 (11.1)</td>
</tr>
<tr>
<td><strong>Developers who support having different names for similar patterns.</strong> (N=22)</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>21 (70.0)</td>
</tr>
<tr>
<td>Yes</td>
<td>9 (30.0)</td>
</tr>
<tr>
<td><strong>Do you support standard documentation practices for SPs? (N=22)</strong></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>1 (3.0)</td>
</tr>
<tr>
<td>Yes</td>
<td>21 (63.6)</td>
</tr>
<tr>
<td>Not sure</td>
<td>1 (3.0)</td>
</tr>
<tr>
<td>Perhaps</td>
<td>10 (30.3)</td>
</tr>
</tbody>
</table>

consideration QAs before utilising SPs, it was found out that about 61% did. The remaining 40% who did not take this factor into consideration raises a question about the quality of developed software. In
case the developers wanted to combine many patterns in the development of a complex subsystem or system without any understanding or consideration of the QAs, the result would be a system that does not meet the stakeholder requirements.

Having used SPs in the past, respondents were asked about their willingness to use them in the future. In response to that question, only 27% are willing to continue using patterns, while the rest declined. Most of the comments given by participants to this question regarding their desire to stop using SPs revolved around the lack of better documentation and suitability to the problem area.

According to the results shown in Table 4.10, proper documentation (21.7%) is one of the better solutions to minimise the factors that discourage or hinder the use of SPs in software development. Introducing a proper framework to guide developers to create, document, evaluate, and use SPs can solve this problem. This is one of the reasons that motivated this study, with the hope of laying down common factors and guidelines about SPs utilisation. A substantial portion of the study was therefore dedicated to analysing the existing references and documentation of SPs as an effort to capture ways in which this process can be improved. Further analysis indicated that most of the known SPs references are GoF book, and POSA team series.

In response to the question regarding current references and standardised process during pattern descriptions, there was an almost split opinion on whether existing references describe a standardised process to create and explain patterns. However, the majority disagreed with the statement. This reveals an existing problem. One participant’s comment was “References seem not to have clearly standardised a way of creating and describing patterns”. Solving this problem could be one of the ways to ensure that more developers will be brought on-board to use SPs during their work and producing systems with required characteristics.

In regard to the references provided at Q10 identifying and stating the relationship between QAs and patterns clearly, large portions of the respondents were not sure. Only 24% of the respondents agreed with this statement, while 17% disagreed. Thus, the existing documentation may not be sufficient for identifying these relationships as explained in Chapter 3. As a result, I recommend further research focusing on the relationship between SPs and QAs.

Additionally, a majority of the respondents were not sure whether these relationships have been proven scientifically or not. The reason for this significant number of respondents who were not sure about the statement may have contributed to the developers’ ignorance about considering QAs before/or after choosing a SP. This is a further revelation that there is a need to improve the existing references and documentation about SPs to be more comprehensive and clear.

It is also of paramount importance to have consistency in all references for SPs. Only 11% of the respondents identified conflicting views in references regarding some SPs and/or QAs. Therefore, such inconsistencies are minimal in pattern documentation. However, it could also be due to the fact that many of the respondents hadn’t taken a closer look into existing references regarding SPs and QAs relationships. Also, it is possible the reason is because of the lack of availability/clarity of such references. Nevertheless, such inconsistencies existed and could cost a lot of time and money. Thus, it is necessary to identify all the inconsistencies within the references and deal with them, or to create a framework that guides developers during their creation and documentation of patterns to produce consistent documentation as explained earlier in this section. There are no existing studies for SPs-QAs relationships, except for what has been reported within this dissertation.
4.4. FINDINGS AND RECOMMENDATIONS

Possibly due to the disadvantages arising from the inconsistencies above, almost two thirds of respondents did not support having different names for similar patterns, deeming it to be more confusing. However, a third of the respondents supported the idea, giving the reason that it would be necessary, especially if the patterns are similar but not identical, and in the case of different contexts of application. Therefore, as in Chapter 3 and to avoid confusion, it is suggested that the variants should be extensions to the main pattern name for ease of identification, such as Proxy-Remote Pattern or Proxy-Virtual Pattern.

One objective of this study is to support standard documentation for SPs, in order to increase their utilisation and ease their evaluation, with a flow-on effect being to ease the evaluation of software architecture. Another benefit is to improve the integration of SPs into other existing components, by eliminating one of the hindrances to their utilisation. 64% of respondents support the idea of having standardised documentation practice for SPs, in line with the objectives of this study. Only one respondent was against standard documentation but with no sufficient reasons or comments provided.

This thesis is a direct contribution towards developing systematic and standardised documentation of SPs. I recommend more research into similar and related fields to document their findings and recommendations as part of the process of achieving this objective.

Following the recommendation to embrace standard documentation practices; four statements (with proposed solutions) were suggested by this study and presented on a Likert scale to the respondents in order to determine how strongly they agree with the statements. The objectives were to firstly determine the opinion of the respondents regarding each statement, and Secondly, to clarify the action points to guide the way forward developing a SPs platform. The four statements are illustrated in the Figure 4.3.

All four statements essentially propose solutions that could help to overcome some of the issues identified in this section. The overlapping (95% CI) error bars on the “Agree = 4” option indicate a strong agreement by the respondents to all four statements, which support this research effort.

Figure 4.3: The level of participants’ agreement to the four statements (proposed solutions).

4.4.1.2 Two Dimensional Analysis

In this sub-section, a two dimensional analysis is reported, in order to uncover the relationship between responses to different pairs of questions. This can reveal a trend or pattern in the responses, which in turn reveals the need to focus on some aspects of the study or balance the dependency between different
factors based on the analysis results and the research objectives. This was the purpose of performing this type of analysis in which every set of responses was analysed in comparison with another set of responses to try to unearth any meaningful relationships. The selection of the question combinations is based on the research goal and objectives.

To start off, the relationship between how frequently SPs are used by developers and their field of expertise within software development is analysed by developing a null and alternative hypothesis. The first variable of the hypothesis represents the mean number of times that developers used styles or patterns in their work with expertise in a specific domain, while the second variable of the hypothesis is the mean number of times the developers used SPs eventhough they had no expertise in software development in that particular domain. Thus, the hypothesis was set as follows:

**Null hypothesis,** $H_0$: $\mu_1 \leq \mu_2$

**Alternative hypothesis,** $H_a$: $\mu_2 > \mu_1$

Where, $\mu_1$ is the mean number of times developers used software styles/patterns during their work with the expertise in a specific domain (e.g., Coding), and $\mu_2$ is the mean number of times they used software styles/patterns during their work if they don’t have expertise in the same domain (e.g., Design). The results are summarised in Figure 4.4 and Table 4.11.

![Figure 4.4: Mean response of how often developers' uses software styles/patterns during their work by their general field of expertise.](image)

Six fields of expertise were considered and analysed individually against the hypothesis. Out of the six, only two of the fields produced significant results. A Levene’s test and t-test were performed (Table 4.11) to show the difference in the usage of SPs between respondents who had expertise in architecture and those who didn’t. Initially, Levene’s F-test confirmed an equal variance between the two variables. Further, an independent sample t-test was performed to statistically verify the inequality of the means between the two variables. The test produced significant results at a 5% level, with $t = 3.64$. This indicated that developers whose expertise lay in architecture utilised SPs more than developers who didn’t have expertise in architecture.

Another field that produced significant results is design as illustrated in Table 4.11. Between two
hypothesised variables with equal variance as indicated by Levene’s F-test, there was a significant difference in the means of SPs utilisation between respondents who had expertise in design and those who didn’t.

Table 4.11: Independent sample t-test for equality of two population group means of "how often developers used software styles/patterns during their work" by developer’s field of expertise.

<table>
<thead>
<tr>
<th>Field of expertise</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Levene’s Test for Equality of Variances: F (p-value)</th>
<th>t-test for Equality of Means t-statistic (p-value)</th>
<th>95% CI of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements elicitation/st</td>
<td>Yes</td>
<td>23</td>
<td>2.13</td>
<td>0.76</td>
<td>5.05 (.029)</td>
<td>-1.27 (.105)</td>
</tr>
<tr>
<td>modelling/analysis</td>
<td>No</td>
<td>25</td>
<td>2.48</td>
<td>1.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project management</td>
<td>Yes</td>
<td>23</td>
<td>2.43</td>
<td>.79</td>
<td>2.03 (.161)</td>
<td>0.83 (.204)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>25</td>
<td>2.20</td>
<td>1.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>Yes</td>
<td>20</td>
<td>2.85</td>
<td>0.88</td>
<td>0.10 (.749)</td>
<td>3.64 (.001)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>28</td>
<td>2.93</td>
<td>0.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>Yes</td>
<td>20</td>
<td>2.57</td>
<td>0.92</td>
<td>0.58 (.450)</td>
<td>2.28 (.014)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>28</td>
<td>1.95</td>
<td>0.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>Yes</td>
<td>33</td>
<td>2.45</td>
<td>1.03</td>
<td>3.95 (.053)</td>
<td>1.52 (.067)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>20</td>
<td>2.00</td>
<td>0.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td>Yes</td>
<td>22</td>
<td>2.41</td>
<td>0.85</td>
<td>1.17 (.284)</td>
<td>0.63 (.266)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>26</td>
<td>2.23</td>
<td>1.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Yes</td>
<td>12</td>
<td>2.58</td>
<td>0.79</td>
<td>0.94 (.337)</td>
<td>1.12 (.135)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>36</td>
<td>2.22</td>
<td>1.02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Similar analyses of the hypothesis for the other fields of expertise, including requirements elicitation, project management, coding, testing, and documentation, all yielded non-significant results, indicating that there was no significant difference in the utilisation of SPs by respondents with expertise in these fields, as compared to those who did not possess expertise in the particular field.

According to the findings above, there is a clear deduction that software styles/patterns are utilised mostly by respondents whose expertise lies in architecture and design, one logical reason is the closeness in nature between both fields. This makes sense since for some (software) fields patterns may not yet exist, such as requirement elicitation. An Architect’s motivation is exploration and discovery of new ideas that can make their designs unique. For this challenge, they may be willing to try even untested ideas and tools. This could be the possible explanation why SPs have been embraced by architects and designers more than other categories of developers, or maybe because of the availability of the large number of patterns in their arena.

On the other hand, experts in other fields such as coding, testing and documentation have a bias towards using well-known and well-tested technologies. They may not be known to try out new tech-
nologies, especially those of which they are not sure, or maybe the natural facts, tools, and processes of their areas constraint the creation and utilisation of patterns. This could explain their reluctance towards utilising SPs in their work. This realisation is another indication that to many developers and experts, the use of SPs is still a foreign idea and more needs to be done to promote the innovation for new SPs or to utilise the existing ones if applicable in their fields of expertise.

SPs and styles were developed to be used in all fields of software development based on context and applicability. According to the 2 dimensional analyses above, it is clear that there could be a bias regarding the utilisation of SPs across different fields of software development. This may not be desirable and raise the question as to whether every type of developer requires knowledge about SPs? Indeed, should they have this knowledge during their work, if and only if, it’s available can improve their missions?

This finding could also raise an alarm about a bias that may arise among some developers who may be inclined to think that patterns and styles are actually meant only for those whose expertise is in architecture and design. However, it should be emphasised that the utilisation of SPs and styles may not lie in just one field of expertise, but rather it is the vehicle that should be utilised in all fields of software development where applicable and efficient. One solution is to promote awareness of SPs and encourage their use/relevance in other parts of software development. As a consequence, interdisciplinary knowledge share will be promoted.

A further analysis of the proposed solutions on questions 17-20 of the survey revealed a variation in the response from respondents according to their field of expertise, as shown in Table D.2. These questions were generally aimed at drawing a conclusion on what should be done to improve the utilisation of SPs. According to this analysis, the strongest agreement to the four statements was from respondents whose expertise lay in “requirements elicitation/modelling/analysis”. On the other hand, the lowest mean of agreement responses was from those with expertise in documentation. Apart from documentation, the mean responses from the other respondents towards all statements were significantly higher than 3% at a 1% level of significance. The analysis showed most of the experts in documentation either agreeing or remaining neutral about the suggestions given in the statements.

The above realisation further raised a question about the need of SPs documentation to those developers whose expertise is documentation. Developers from each software field have their own requirements and tools, which facilitate their work. However, the nature of each field varies, so their need for SPs and pattern suitability varies among them, too. It would be expected that in order for respondents to give their opinion in terms of agreeing or disagreeing with the suggested questions aimed at improving the utilisation of SPs in software development, they would need to be well furnished with information about systems and SPs. Based on this, it can therefore be concluded that most of the experts in documentation did not have much experience with patterns. This could explain why they decided to remain neutral to the statements. It can therefore be concluded that SPs may be less applicable in the field of documentation.

Despite the discrepancies above in the agreement to the four statements by respondents from different fields of expertise, it was generally noticed that there was agreement to the four statements. Based on this finding, the proposition that it is vital to increase the use of software patterns in software development is confirmed. Thus, this research has proposed and developed (see Chapter 3) a solution, which could contribute towards improving the experience of using SPs in software development, and the findings prove the validity of the proposition across all fields of software development. I recommend that all the suggestions recommended in these statements to be put into consideration for further research and
eventual implementation.

It was necessary to study the relationship of the respondents’ agreement between the statements and the sector in which developers gained their experience. The detailed results of this analysis are illustrated in Figure 4.5, while more analysis can be found in the related appendix, Section D.3.3.

Figure 4.5: Mean plots (a) to (d) for each of the statements (Q17 to Q20) in Y-axis, along with developers’ work sectors in X-axis. (A=Academia; I=Industry; G=Government).

For the first statement (Q17), Figure (a), overall mean agreement was 4.1, where the academic sector received the highest mean agreement of 4.5, then industry, and government, respectively.

The next statement (Q18), Figure (b), overall mean agreement of the respondents was 3.7, where the highest mean response was from the academic sector, then the industry sector, followed by the government sector.

The Figure (c) statement (Q19) received an overall mean agreement of 4.1, where the academic sector showed their highest mean agreement of 4.5, followed by the respondents from industry (4.0) and then government (3.8).

Finally, there was an overall mean agreement of 3.9 to the statement (Q20) in Figure (d), where the respondents from the government sector had the highest mean agreement of 4.0, while the respondents of academic sector had the least mean agreement of 3.9.
A general analysis of these results indicates that the highest rate of agreement to the first three statements came from developers who had gained most of their experience in the academic sector, followed by industry and lastly government, while the agreement to the last statement is vice versa. In a statistical sense, as illustrated in Table D.4—(Section D.3.3), all of the statements’ population mean for all sectors are significantly higher than 3 (neutral value) at 1% level of significance.

A sensible explanation to this result is: based on propositions aimed at researching an existing problem and analysing possible solutions. The academic sector, being more research-oriented, is expected to agree with the statements more, since the statements support their efforts to do research and analysis. The rate of agreement is followed by developers in the industry sector, which makes sense again, since developers in the industry sector are aimed at solving current problems by analysing and doing extensive research, then recommending or experimenting with solutions to the problem domain. The government sector, which had the lowest mean agreement for the first three statements, is not known to be research-oriented. Their main task is to sustain the existing quality and ensuring that the systems they use are stable. This explains the differences in the mean count of the total agreements.

However, the last statement presents a unique situation in which developers from the government sector had the highest mean agreement, followed by academic then industry. The last statement proposes the development of an evaluation model to be used to solve the problem domain, partially. It could be expected that developers from the academic sector would support this more. On the contrary, they showed less support for the statement, which lacks sense, because standardisation is a better approach and is supported by many references and researchers, as explained in Chapter 3.

SPs possess different QAs. As a result, as stated earlier, the examination of QAs is of paramount importance before deciding to utilise a software pattern. Thus, the relationship between whether respondents were aware of patterns and their agreement to the statements in Question 17–20, as shown in Figure 4.6 and Table D.5.

![Error bars: 95 %CI](developing_an_evaluation_model_to_assess_patterns_against_quality_attributes_is_important_.png)

Figure 4.6: The mean of the items for developers who are aware of software styles/patterns.

The bar chart in Figure 4.6 shows that all statements mean are significantly higher than 3 (the neutral value) for those who were aware of SPs. The error bars (95% CI) show that all of the mean responses are not statistically different from each other, as their error bars overlap each other. This is an interesting result, as it would be expected that those who were not aware of SPs would either disagree
with the propositions or decline to respond. Contrary to this, all respondents generally agreed with the statements regardless of their awareness due to the sound of the proposed solutions, within the four statements. Therefore, further research is recommended.

A significant relationship was also noted between the frequency use of SPs by developers and their care for QAs during their utilisation of patterns as represented in Table 4.12. A group descriptive statistic and independent sample t-tests were conducted to determine the equality of the two means. They showed that there is a significant difference between the frequency of use of SPs by developers who took into consideration QAs during their selection of patterns as compared to those who didn’t.

Table 4.12: Independent sample t-test for equality of two population group means of “how often developers used software styles/patterns during their work”.

<table>
<thead>
<tr>
<th>Agreement (Yes/No)</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Levene’s Test for Equality of Variances: F (p-value)</th>
<th>t-test for Equality of Means</th>
<th>95% CI of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>t-statistic (p-value)</td>
<td></td>
</tr>
<tr>
<td>Developers who care about or consider quality attributes during selection of patterns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.51 (.123)</td>
<td>2.22 (.017)</td>
</tr>
<tr>
<td>Yes</td>
<td>22</td>
<td>2.73</td>
<td>0.63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>14</td>
<td>2.14</td>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If software patterns used in the past, will keep using them in the future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.00 (1.000)</td>
<td>5.26 (.000)</td>
</tr>
<tr>
<td>Yes</td>
<td>27</td>
<td>2.93</td>
<td>0.73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>10</td>
<td>1.60</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identified any conflicting views among any references regarding some patterns and/or regarding quality attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>3</td>
<td>3.00</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>24</td>
<td>2.50</td>
<td>0.78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support having different names for similar patterns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>9</td>
<td>2.44</td>
<td>0.88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>21</td>
<td>2.76</td>
<td>0.89</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The deduction from the above analysis is that developers who cared to consider QAs before selecting SPs used the patterns more frequently than those who did not. A logical explanation beyond this is, as developers start using patterns, they are only interested in the basic use of SPs. This changes, as the developers continue using SPs because developers are likely to develop more interest in knowing the features and characteristics of patterns to make their utilisation more efficient. Hence, the more frequent use of patterns by developers, the more careful they are in considering their QAs before selecting them.

As noted earlier, the consideration of QAs when selecting SPs is very beneficial to a developer. The result of this is the production of improved quality systems and software. This research therefore recommends SP stakeholders to adapt or develop methods that will inform them about the output quality of a system or software before implementing required pattern.

The last significant result in this section is the relationship between developers who used SPs in the past and will continue using SPs in the future, and the developers’ frequency of use of SPs, as shown in Table 4.12. According to the result, there is a significant difference in the frequency of use of SPs between respondents who had used patterns and were willing to continue using them in the future, and those who hadn’t. Therefore the frequency of use of SPs is higher among developers who had used SPs and were willing to continue using them in the future.

This finding was anticipated. The frequency of usage of SPs can be directly linked to the comfort
or ease with which developers use them. *It is obvious that the more a developer has spent time using SPs, the more comfortable the developer becomes with them.* As a result of this, the developer is likely to use patterns more frequently. Whereas, developers who are not using patterns frequently could have their reasons based on their needs and how comfortable they are with patterns. Thus, it is less likely that they will continue using patterns in the future.

### 4.4.2 Important comments from some non-significant results that related to the research goals

In this section some non-significant results worthy of comment are considered.

- The analysis between the relationship of the respondents’ work place (sectors) and how frequently they used patterns is depicted in Figure 4.7.

![Figure 4.7: Developers’ work sectors vs how often they used software patterns (frequency in stacked bar chart).](image)

The analysis concludes that respondents whose expertise lies in academia used SPs less frequently than those in government and industry. However, this relationship was not significant, according to the Chi-square test, Section D.3.2. As such, there is no statistical evidence to support this relationship. Nevertheless, this is an indication of yet another discrepancy in the embracement of patterns in different sectors of the economy. It raises a question about what could cause this discrepancy? In order to understand this, further research is needed to reveal some of the factors that could influence embracement and utilisation of SPs in different sectors of the economy.

- Other analysis, as illustrated in Table 4.13, revealed that the number of years of experience in software development did not seem to affect the respondents’ decision to be in favour of having different names for the same pattern, support for SPs standardised documentation, or the agreement with the statements about the important knowledge that needs to be possessed by software engineers and developers regarding SPs-QAs relationships.

- Also, there could be a possibility that the variation in SPs usage with different sectors is affected by the approach taken by developers when using SPs. In order to use the patterns effectively and with maximum satisfaction, it is necessary to consider their QAs. Also, to establish if this possibility
exists, a two dimensional analysis was performed to determine the relationship between the sectors of economy in which the respondent developers gained most of their development experience, and whether they considered QAs during their selection of patterns. The analysis is reported in Table D.3.

Table 4.13: Cross tabulation of three work sectors with software styles/patterns usage and consideration of quality attributes and Chi-square test of independence between attributes.

<table>
<thead>
<tr>
<th>Are you aware of software styles / patterns?</th>
<th>How many years experience do you have in total in the software development field?</th>
<th>Chi-square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>5-10 (years)</td>
<td>10-15 (years)</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Are you aware of software styles / patterns?</td>
<td>How often do you use software styles / patterns during your work?</td>
<td>Chi-square (p-value)</td>
</tr>
<tr>
<td>Yes</td>
<td>Never</td>
<td>Infrequently (&lt;10%)</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>No</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Have you identified any conflicting views among any references regarding some patterns and/or regarding quality attributes?</td>
<td>Chi-square (p-value)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>No</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Do you support having different names for similar patterns?</td>
<td>Chi-square (p-value)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>Do you support standard documentation practices for software patterns?</td>
<td>Chi-square (p-value)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>No</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Not sure</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Perhaps</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

The results of the study showed that an equal number of developers from academic and industry consider the quality attributes of software patterns before utilisation, while less of those in government cared. However, this relationship was not clearly defined according to the Chi-square test.

• Another important two dimensional analysis was between the number of years of experience that respondents had, and their awareness about software patterns. The results of the analysis are shown in Table 4.13 and Figure D.2. Out of the 41 respondents to the question, 61% of them had an experience of 5 to 10 years; and 78% of whom were aware of SPs. On the other hand, 89% of those
with an experience of over 10 years were aware of SPs.

Thus, a **positive relationship** was identified here, *the more experienced the respondents were, the more likely they were to be aware of software patterns*. However, the Chi-square test indicated that the dependency on awareness about SPs on the year of experience in software development was non-significant. Although preliminary matrix analysis tended to show that most of the respondents who used patterns in development had less than 15 years of experience, an hypothesised Chi-square test ruled out the dependency of SPs usage on the years of experience. In the same way, it was noticed that the identification of conflicting views within the documentation of SPs was not dependent on the years of experience in software development. This makes sense since, as stated earlier in this chapter, all the participants in the questionnaire were suitable for the research since the length of experience as short as 5 years was enough for the respondents to know about SPs. Since almost all the respondents had this experience, the difference in awareness about patterns was expected to be non-significant, proving once again that the choice of respondents in this survey was done effectively and the responses to the key questions could be relied upon.

### 4.4.3 Inappropriateness of three and four dimensional analyses

As indicated earlier, 3 and 4 dimensional analysis of the responses in this questionnaire proved to be non-significant. In order to prove this, an example of 3 and 4 dimensional analysis is illustrated in Section D.3.6. This type of analysis provided a null difference among all the responses given. Based on this, it was therefore clear that 2 dimensional analyses were sufficient to exhaust all the underlying relationships between responses to different questions.

### 4.5 Related work

The use of SPs and Styles is one of the many aspects of software design. There are many other design aspects and approaches that have been surveyed by various researchers in order to improve the overall process of software design. Most of these studies focus on particular views of architecture or design, such as the survey of architecture design rationale by Tang et al. [2006] that sought to discover and emphasise the importance of systematic documentation in software design. Their research was aimed at finding out the significance of using and capturing the design rationale among software practitioners. Also, the work done by Malavolta et al. [2013] is noticed, which aimed at establishing the needs of architectural languages in industry.

In addition, the effort by Zhang et al. [2013] to experience user perceptions about software design patterns is a positive contribution towards SPs in particular. The survey gathered respondents answers to the question: “which design patterns from the GoF do expert pattern users consider as useful or not useful for software development and maintenance, and why?”.

Also, the design of empirical studies to evaluate software patterns, such as a survey by Abou-El-Fittouh et al. [2012], which summarises the study designs of software patterns available in the literature and includes evaluation criteria.

The two former works above (Tang et al. [2006] and Zhang et al. [2013]), are among a number of others that have been conducted with the view to improving the process of software design and architec-
4.6. LIMITATIONS

ture. However, none of them narrowed down to focus on these study aspects to encourage development of a systematic solution to document SPs better. The latter two surveys were too narrowed and specific to the GoF design patterns, and an investigation of the existing patterns studies, respectively.

Therefore this survey is important and can be seen as a proper combination of the two approaches, because it’s not too specific or too wide. Also, it is concentrated on SPs in general and from different aspects, such as, pattern documentation, factors deterring or supporting the use of patterns, and their relationships with QAs.

Also, good research has been conducted on “software design-pattern specification languages”, Khwaja et al. [2016]. Their focus was on object-oriented design patterns. Their research discussed languages based on, mathematical formalism, other modelling languages, and other languages. They did categorise them into description, analysis, and detection patterns languages. Despite the variation between the objectives and scope of this research and their survey, one common conclusion was “the need for a unified framework for the evaluation of design patterns across different domains”.

To sum up this section, the current research discussed, including this survey, have a common goal to improve the process of software design and architecture (including its styles/patterns). Thus, I recommend inclusion of the findings of this current research into the process of developing concrete solutions for current software architecture and design challenges.

4.6 Limitations

It is worthy to note that this research has its own limitations, which calls for more research within related fields to fill the gaps identified in the research. The first limitation of this study is the optional questions, which allow the participants to skip some of the questions without providing answers. This results in low cell responses in the three and four dimensional analyses (explained earlier in Section 4.3.5, which leads to some invalid results, and minimises the chances of finding new significant relationships between different factors that could improve pattern utilisation. Another limitation is that most questions included in the questionnaire were close-ended. This may limit the opinions gathered from the respondents. Also, the limited sample size caused some invalid analysis results, which consequently limits the findings of this work.

Finally, this chapter only focused on SPs without considering the possibility of a better alternative to SPs for software designing and development. Therefore, I recommend more studies, as opposed to relying solely on the findings of this research, in order to come-up with solutions for some of the issues pointed out here.

4.7 Conclusion

The utilisation of well-documented SPs has the potential to improve the whole process of software evaluation and development, especially in architectural and design levels. However, in order for this potential to be realized, there is a need to identify the factors that could be hindering the use of SPs and come up with solutions. Research is one of the best tools to identify such factors and facilitate the proposition of possible solutions.

Through an inductive approach, this research has made an effort to establish the current trends in the utilisation SPs within the software development process. The research was able to capture a wide
audience across various fields of software development and collect their opinions about the factors that encourage or discourage the utilisation of software style/patterns. It was established that one of the main factors influencing the utilisation of SPs is documentation. This is also likely to affect how patterns will be utilised in the future. This is the reason why part of the thesis was dedicated towards analysing current and reliable patterns’ documentation, which aimed to help in establishing a systematic framework to provide clear guidelines for creating and documenting SPs.

Based on the findings of this study, the importance of QAs consideration within SPs cannot be ignored, as shown by the first statement in Table 4.14. Thus, software developers who consider QAs before using patterns are likely to be able to build better systems. It can therefore be concluded that the effectiveness with which software patterns can be applied depends largely on their QAs.

From the study, it was also clear that the continued utilisation of patterns in the future depends largely on the approach of their documentation. Findings also show that the respondents agreed to the suggested solutions, which were provided partially through statements in Q17 to Q20 of the survey. I therefore recommend that the suggestions be embraced and more research be conducted on each of them to determine the best way in which they can be implemented, in order to develop a SPs platform.

Table 4.14: Summary of the main significant analysis results with associated tables (for better traceability)

<table>
<thead>
<tr>
<th>Findings</th>
<th>Analysis Table</th>
<th>Statistics (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developers significantly agreed with the statement “Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field”</td>
<td>4.10</td>
<td>t = 9.50 (.0000)</td>
</tr>
<tr>
<td>Developers significantly agreed with the statement “Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess”</td>
<td>4.10</td>
<td>t = 4.34 (.0000)</td>
</tr>
<tr>
<td>Developers significantly agreed with the statement “Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardised quality attribute definitions, is valuable knowledge”</td>
<td>4.10</td>
<td>t = 6.34 (.0000)</td>
</tr>
<tr>
<td>Developers significantly agreed with the statement “Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use”</td>
<td>4.10</td>
<td>t = 5.78 (.0000)</td>
</tr>
<tr>
<td>The developers who have expertise in architecture more frequently used software styles/patterns during their work, compared to who don’t have the expertise in architecture</td>
<td>4.11</td>
<td>t = 3.64 (.0005)</td>
</tr>
<tr>
<td>Designers more frequently used software styles/patterns during their work compared to the developers who have other expertise.</td>
<td>4.11</td>
<td>t = 2.28 (.0070)</td>
</tr>
<tr>
<td>Programmers (Coders) more frequently used software styles/patterns during their work compared to the developers who have other expertise.</td>
<td>4.11</td>
<td>t = 1.52 (.0335)</td>
</tr>
</tbody>
</table>
4.7. CONCLUSION

|developers who “care about or consider quality of attributes during their selection of patterns” used software styles/patterns more frequently than who did not care | 4.12 | \( t = 2.22 \) \( (0.0085) \) |
| developers who “used software styles/patterns in the past and will keep using them in the future”, use patterns more frequently | 4.12 | \( t = 5.26 \) \( (0.0000) \) |

In conclusion, SA and design can be greatly improved through the application of SPs. However, this process can also be hindered if software developers do not have the right experience and guidelines to apply SPs correctly. This research has laid out some important observations regarding the current trends in SPs. Further research is required as recommended within the findings section in order to ascertain how each of the issues discovered can be handled. In the end, through this study and other related research, a systematic framework can be achieved to guide the seamless application of software patterns in various fields. Two of the most urgent areas that need further research as recommended in this study include Documentation and QAs of SPs. The anticipated result of successful application of SPs in software development is increased as developers improve the quality of their software, as well as saving time in the development process. Table 4.14 summarises the significant findings of this chapter.
Utilisation of Software Architecture Artefacts and its Evaluation

The architecture of complex software or systems is a collection of hard decisions that are very expensive to change. Successful product development and evolution depend on making the right architectural choices. Can you afford not to identify and not to evaluate these choices?

—Alexander Ran, Principal Scientist of Software Architecture, Noki

Clements et al. [2002a, back cover]
5.1 Executive summary

The quality of a software product depends largely on the architecture and design processes. Quality characteristics are measured and evaluated using various evaluation methods. Also, the utilisation of various software architecture artefacts has *NOT* been fully embraced by software developers, as noted from the previous questionnaire analysis in Chapter 4. This raised curiosity about the factors that could hinder or encourage the utilisation of these artefacts. Therefore, this research reported in this chapter is a continuing effort to the study reported in previous chapters.

Thus, **one of the main aims** in this chapter is to reveal the various factors that affect positively or negatively SA model/artefact utilisation among professionals from various fields; **the second aim** is to uncover any relationships between Software Architecture Evaluation (SAE) and various factors affecting SA; in order to improve the supportive factors and decrease the impact of the hindering factors. The research methods used during this part of the research were a **survey and field study**.

5.2 Introduction

Having realised the importance of SPs in Chapter 4, which form an essential part of SA artefacts, it seems apparent that there is a need for more research regarding SA and SAE, in order to improved their methodologies. Despite the issues concerning SP-QA relationships, the analysis in Chapter 4 revealed that there is a high potential for quality improvement during the proper utilisation of SPs, with a consequential improving of SA descriptions and assessment.

There are various studies that have tried to analyse the relationship between SA and QAs. Such studies include the research by Bass et al. [1998], Moreno et al. [2008], Zayaraz [2010], AV Sriharsha [2015], Khwaja et al. [2016], Lugou et al. [2016], Ribeiro et al. [2017], and Brouwers et al. [2017]. Each of these studies focused on a specific architecture view, such as SPs, a specific QA, or modelling and languages.

However, there is no study yet published, which combines the investigation of SA, SAE, and SA modelling techniques, through two different approaches, as is case in this study. The two pillars of investigation (brown boxes) reported in this chapter are illustrated in Figure 5.1.

![Chapter-5 Main Sections](image)

Figure 5.1: The two pillars to investigate SA, SAE, and SPs relationships.

Therefore, the objectives of this chapter are:

- To explore the current trend of SA development among developers from different fields of software development. By studying these trends, the courses of action (e.g. more...
teaching of SA in academia) within the fields of software development should be able to be determined.

- To determine the factors engaging/discouraging the utilisation of modelling techniques during SA development.
- To disclose the challenging factors that are deterring overall SAE methods’ progress, and quantitative assessment in particular.
- To uncover the important factors that determine or influence the utilisation of software architecture artefacts.
- To study SA and SAE overall trends, in the government and industry sectors, through a field study.

Both methods (survey and field study) are utilised to obtain, observe, and analyse information from different organisations and from a varied range of software developers, in order to draw more concrete judgements and recommendations. Both methods and their findings are described in detail in Sections (5.3 and 5.4).

"Such trends include the familiarity of developers with software architecture descriptions, modelling techniques, evaluation processes, their frequent use of these methods, and the future implications concerning the final deliverables.

5.3 Survey methodology and process

The questionnaire for this second survey was sent to the same people and organisations mentioned in Chapter 4, and 50 responses were received (compared with 52 for the first survey).

Mostly the survey methodology, process, and analysis techniques are the same as described in Chapter 4. Thus, this section and its sub-sections will report the differences only, to avoid any repetition. Aspects that will not be addressed here, because there is no difference from Chapter 4, are:

- Research technique and process.
- Invitation mechanism and instrument distribution.
- Target population and sampling technique.
- Rationale of the selected analysis methods.

5.3.1 Instrument questions

The survey is composed of twenty-three questions, that are divided into three main sections, as illustrated in Figure 5.2.
The major part of the survey focuses on five elements that have an affect on SA, and consequently on SAE. These elements were selected due to their high influence and relationship to both domains (as illustrated in Figure 5.3), as determined from work experience and deep investigation of the state of the art.

Each section of the survey has its own objectives that serve the overall survey goal as described in Tables (5.1, 5.2, and 5.3), respectively.

Table 5.1: Section 1. Background Questions (Questions marked with (*) are mandatory).

<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1*</td>
<td>What is your general field of expertise regarding software development? You may select more than one of the following options.</td>
</tr>
<tr>
<td></td>
<td>- Requirements elicitation / modelling /analysis</td>
</tr>
<tr>
<td></td>
<td>- Project management</td>
</tr>
<tr>
<td></td>
<td>- Architecture</td>
</tr>
<tr>
<td></td>
<td>- Design</td>
</tr>
<tr>
<td></td>
<td>- Coding</td>
</tr>
<tr>
<td></td>
<td>- Testing</td>
</tr>
<tr>
<td></td>
<td>- Documentation</td>
</tr>
<tr>
<td></td>
<td>- Other - please specify:</td>
</tr>
<tr>
<td>Q2*</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
</tr>
<tr>
<td></td>
<td>- Academia</td>
</tr>
<tr>
<td></td>
<td>- Industry</td>
</tr>
<tr>
<td></td>
<td>- Government</td>
</tr>
</tbody>
</table>
### 5.3. Survey Methodology and Process

<table>
<thead>
<tr>
<th>Q3*</th>
<th>How many years’ experience do you have in total in the software/systems development field?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ 5–10 (years)</td>
</tr>
<tr>
<td></td>
<td>☐ 10—15 (years)</td>
</tr>
<tr>
<td></td>
<td>☐ 15–20 (years)</td>
</tr>
<tr>
<td></td>
<td>☐ 20–25 (years)</td>
</tr>
<tr>
<td></td>
<td>☐ Over 25 (years)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q4</th>
<th>Are you aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ Yes, which language?</td>
</tr>
<tr>
<td></td>
<td>☐ No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q5</th>
<th>How often do you use models to describe software/system architecture during your work?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ Never</td>
</tr>
<tr>
<td></td>
<td>☐ Infrequently (&lt;10%)</td>
</tr>
<tr>
<td></td>
<td>☐ Reasonably frequently ( &gt; 15 % and &lt; 50 %)</td>
</tr>
<tr>
<td></td>
<td>☐ Regularly (&gt; 50 % and &lt; 80 %)</td>
</tr>
<tr>
<td></td>
<td>☐ Nearly always (&gt; 90 %)</td>
</tr>
</tbody>
</table>

Table 5.2: Section 2. Software architecture utilisation questions.

This section is mainly focused on the matters relating to SA utilisation, description, and modelling amongst software developers. Also, it includes the factors that are likely have an affect on the utilisation of SA modelling techniques during development process. The section went further to gather information related to software description languages.
What are the main factors that **ENCOURAGE** the utilization of modelling techniques to describe software/system architecture? You may select up to two of the following options.

- [ ] Easier to demonstrate the software/system concept and features.
- [ ] Most available architecture modelling references are clear and well documented, which helps developers.
- [ ] Understand and apply the modelling approach easily.
- [ ] It makes the designers/programmers job much easier.
- [ ] Makes the evaluation of stakeholders’ requirements for quality attributes possible in the early stages of the development lifecycle.
- [ ] Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier.
- [ ] The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible.
- [ ] Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, UML).
- [ ] Teaching of the architecture modelling languages in the academic sectors.

---

What are the main factors that **DISCOURAGE** the utilization of modelling techniques to describe software/system architecture? You may select up to two options.

- [ ] Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system.
- [ ] Lack of standardisation between existing architecture modelling techniques, notations, and semantics. Current architecture description languages (including modelling languages) are still immature.
- [ ] Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money.
- [ ] Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance).

---

From your experience, what is the best language to use to describe software/system architecture, so as to be more useful to all stakeholders, and to be easier to undertake qualitative and quantitative assessments?

- [ ] Natural language (e.g. English-text) only.
- [ ] Semi-formal language (e.g. UML, SysML) only.
- [ ] Formal language (e.g. ADLs, Z) only.
- [ ] Formal language & Natural language together.
- [ ] Semi-formal language & Natural language together.
- [ ] Semi-formal language & Formal language together.
- [ ] All three languages above together.
5.3. SURVEY METHODOLOGY AND PROCESS

| Q9 | Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture.  
○ Strongly Agree  
○ Agree  
○ Neutral  
○ Disagree  
○ Strongly Disagree |

| Q10 | Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.  
○ Strongly Agree  
○ Agree  
○ Neutral  
○ Disagree  
○ Strongly Disagree |

Table 5.3: Section 3. SA evaluation methods and tools.

The final section consists of thirteen questions, which focused on SAE tactics, and factors that could support or hinder SAE methods. The section also, sought to explore the effect of the current technologies, automation and tools on SAE methodologies.

| Q11 | Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models, (e.g. detecting attacks for security).  
○ Yes, please provide reference:  
○ No |

| Q12 | Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?  
○ Yes, please name it:  
○ No |

| Q13 | What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.  
☐ The language used for describing SA.  
☐ Formality level of SA description.  
☐ Using standard language and architecture framework for describing SA.  
☐ Tools availability for describing and evaluating SA.  
☐ Documenting mechanism used during SA description. |

169
### Questions and Answers

**Q14**

What are the most important factors that could *HINDER* quantitative evaluation for any SA? You may choose two.

- □ The language used for describing SA.
- □ Formality level of SA description.
- □ Using standard language and architecture framework for describing SA.
- □ Tools availability for describing and evaluating SA.
- □ Documenting mechanism used during SA description.

**Q15**

"Architecture is design, but *NOT* all design is Architecture”

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

**Q16**

There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

Please indicate your level of agreement with respect to the following statements:

**Q17**

Most of the existing software architecture evaluation methods, produce qualitative results.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

**Q18**

It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree
### Q19
Reliable tools are important for developing/or evaluating software/system architectures.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### Q20
Current technology lacks reliable software architecture evaluation tools.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### Q21
Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### Q22
Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### Q23
Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes.
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

### 5.3.2 The analyses methods and procedure

A **Dimensional Approach** was used to analyse the questionnaire.

*One Dimensional* (or Individual) analysis - where each question is analysed individually; and *Multi*
Dimensional (or Matrix analysis) - where more than one question is analysed together. Both (Individual and Multi dimensional) analysis follow the same methods and procedure that have been used in Chapter 4. However, the matrix combinations for the Multi dimensional analysis is different. The two dimensional matrices are shown in Table 5.4, whereas, the three and four dimensional matrix tables are not included here due to their invalid results, as explained later in Section 5.3.3.2. Proper statistical analysis methods were applied.

5.3.2.1 Data Distribution Normality statistics

An assumption of normality for the Likert Scale statements, for questions Q9, Q10 and Q15-Q23 was performed using the Shapiro-Wilk W statistic, which confirmed that each of the answers came from a normal population (p-value > .05), except for Q22. Thus, parametric tests are applicable to all the answers except for Q22. For this question, W = 0.92, Z = 2.61, p-value < .01 but since the Shapiro-Wilk W statistic is close to 1, parametric tests are considered to still be applicable for Q22.

5.3.3 Findings and Recommendations

Having taken into consideration, all the controls explained above for the survey analysis, the responses were gathered from participants, analysed, and documented. The findings portray interesting trends for SA descriptions and SAE, most of which are significant and therefore help in attaining this study’s objectives. To avoid confusion, Figure 5.4 illustrates the organisation of the analysis sections.

![Figure 5.4: The three main sections and their sub-sections of the analysis.](image)

![Figure 5.5: Respondent’s general field of expertise regarding software development.](image)

172
5.3. SURVEY METHODOLOGY AND PROCESS

Demographic data is illustrated through the bar chart in Figure 5.5 and Table 5.5, which summarise the results of the background questions. The responses were received from the same countries shown in Table 4.5. The first step of this analysis was to ensure that all the basic requirements for the survey were met. That is a correct selection of respondents, in terms of years of experience, area of expertise, the sector in which they have worked, as well as their familiarity with the research domain. This was the aim of the first section of the questionnaire, which was analysed individually.

The combination of 31 designers, 20 architects, 23 analysts, and 29 programmers (Coding) is considered a reasonable diversity of 'areas' within the software engineering field from which to obtain responses, and which in turn could lead to credible findings, noting that one participant can select more than one role. The analysis was done according to the actual number of respondents, not how many roles they have occupied during their career; however, the sum of each role was valuable during the comparison process between the respondent’s comments and their fields of experience.

The results indicate that the intent to include experts from most software development fields, in order to report this study across a wide range of disciplines, has been met.
Table 5.4: Two dimensional matrices analysis (only two questions will be analysed together).

| Q | 2   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 | X_1 |     | X_9 |     |     |     |     | X_{14} |     | X_{16} |     |     |     | X_{18} |     |     |     | X_{20} |     |     |     | X_{21} |
| 2 |     |     |     | X_{10} |     | X_{12} |     |     | X_{14} |     |     |     |     |     | X_{18} |     |     |     |     |     |     |     |     |     |
| 3 |     | X_{27} |     | X_{29} |     | X_{31} |     |     |     | X_{35} |     |     |     |     |     |     |     | X_{39} |     |     |     |     |     |     |     |     |
| 4 |     |     |     |     | X_{47} |     | X_{49} |     |     | X_{51} |     |     |     |     |     |     |     |     | X_{55} |     |     |     |     |     |     |     |
| 5 |     |     |     |     |     | X_{56} |     |     |     |     | X_{60} |     |     |     |     |     |     |     |     |     |     | X_{66} |     |     |     |     |
| 6 |     |     |     |     |     |     |     | X_{76} |     |     |     | X_{80} |     |     |     |     |     |     |     |     |     |     |     | X_{83} |     |     |     |
| 7 |     |     |     |     |     |     |     |     | X_{98} |     |     |     |     |     |     |     | X_{96} |     |     |     |     |     |     |     |     |     |
| 8 |     |     |     |     |     |     |     |     |     |     |     | X_{105} |     |     |     |     |     | X_{109} |     |     |     |     |     |     |     |     |
| 9 |     |     |     |     |     |     |     |     |     |     |     |     | X_{110} |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

n in X_n stands for matrix number.

Example: For Question (1), two matrices can be written down as:

- Matrix (X_1): Analysis of (Q1 and Q2) together;
- Matrix (X_7): Analysis of Q1 with each answer for the questions (Q15 to Q23).
An equally significant result involves the sectors in which the participants gained their expertise. The Chi Square ($\chi^2$) test indicates a significant difference between the three sectors at the 5% interval. It is evident from the results, that almost half of the participants gained their experience in academia, followed by industry and government respectively, as shown in Table 5.5. Sector variation could, therefore have an effect on the findings of this survey.

One of the goals of this study was to collect responses from professionals with experience of no less than 5 years. A Chi Square test was used to determine the distribution of these percentages, which appeared to be significant. The biggest percentage of participants lay between 5 to 10 years of experience. However, the distribution of the percentages is NOT uniform, which could possibly influence the responses of the participants in the questionnaire.

Also, with 30% having more than 15 years of experience, this may affect the survey analysis due to the participants familiarity with SA. However, a developer with an experience of 5 years and more is still considered to be able to evaluate SA and design. Thus, the responses collected from these participants can be counted as valuable.

Table 5.5: Frequency distribution of single response categorical questions/items and Chi-square test results for equality of group frequencies; (Q2-Q5, Q8, Q11, Q12).

<table>
<thead>
<tr>
<th>Question/Item</th>
<th>Frequency (%)</th>
<th>$\chi^2$ (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sectors that participants gained most of their general software development experience (N=50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Academia</td>
<td>24 (48.0)</td>
<td>7.84 (.020)</td>
</tr>
<tr>
<td>Industry</td>
<td>18 (36.0)</td>
<td></td>
</tr>
<tr>
<td>Government</td>
<td>8 (16.0)</td>
<td>(.020)</td>
</tr>
<tr>
<td>Total years of experience in the software/systems development field (N=50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5–10 (years)</td>
<td>24 (48.0)</td>
<td></td>
</tr>
<tr>
<td>10–15 (years)</td>
<td>11 (22.0)</td>
<td></td>
</tr>
<tr>
<td>15–20 (years)</td>
<td>4 (8.0)</td>
<td>27.80 (.000)</td>
</tr>
<tr>
<td>20–25 (years)</td>
<td>7 (14.0)</td>
<td>(.000)</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>4 (8.0)</td>
<td></td>
</tr>
<tr>
<td>Awareness of any software/system architectural description/modelling languages, e.g. ADLs, AADL, SysML, UML, etc. (N=49)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>41 (83.7)</td>
<td>22.22 (.000)</td>
</tr>
<tr>
<td>No</td>
<td>8 (16.3)</td>
<td></td>
</tr>
<tr>
<td>How often respondents used models to describe software/system architecture during their work (N=50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Never</td>
<td>7 (14.0)</td>
<td>17.00 (.002)</td>
</tr>
<tr>
<td>Infrequently (&lt; 10 %)</td>
<td>18 (36.0)</td>
<td></td>
</tr>
<tr>
<td>Reasonably frequently (&gt;15% and &lt;50%)</td>
<td>16 (32.0)</td>
<td>(.002)</td>
</tr>
<tr>
<td>Regularly (&gt;50% and &lt;80%)</td>
<td>5 (10.0)</td>
<td></td>
</tr>
<tr>
<td>Nearly always (&gt;90 %)</td>
<td>4 (8.0)</td>
<td></td>
</tr>
<tr>
<td>The best language to use to describe software/system architecture (N=46)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Natural language only</td>
<td>4 (8.7)</td>
<td>60.2 (.000)</td>
</tr>
<tr>
<td>Semi-formal language only</td>
<td>8 (17.4)</td>
<td></td>
</tr>
<tr>
<td>Formal language only</td>
<td>1 (2.2)</td>
<td></td>
</tr>
</tbody>
</table>
### 5.3.3.1 Significant Results

Significant results indicate the existence of a relationship between variables that is not likely to have happened purely by accident. The following sections provide details of the analyses undertaken based on the objectives of this study that serve the research’s main goal.

#### 5.3.3.1.1 One Dimensional Analysis Results

The analysis of question four (Table 5.5), provides a backup to the validity of the results, because a vast majority of the respondents were aware of software/system architectural description and modelling languages, which indicates that 5 years experience is still sufficient to give valid answers to the rest of the questions. Nonetheless, the number of unaware respondents regarding software architectural modelling languages, which includes participants with 15 years of experience, raises a concern. The are potentially many reasons for their unawareness, including poor documentation of SA description and modelling languages, failure to include them in the educational curriculum, they have only ever coded and never having designed a software system, ...etc. The recommendation here for the future is to incorporate the study of such languages into academic syllabuses to promote awareness about them. However, this question gives an important insight into the ‘general usage’ of SA modelling techniques among software developers.

Apart from the awareness regarding SA description languages, 50% of the respondents either used models infrequently or did NOT use them at all when describing SAs. Only a total of 18% used models regularly or nearly always, while another 32% used models reasonably frequently. The distribution of respondents in these categories is significantly non-uniform as determined using the Chi-Square test shown in Table 5.5.

These results are surprising, considering the number of developers who are aware of SA description languages. The reason for infrequent use of models, may therefore NOT be due to lack of their awareness. This factor lays strong emphasis on the need for this study, because the possibility of other (undiscovered) influencing factors that could be discouraging the use of models during SA description apparently exist, and if uncovered by this study will assist in determining the best solutions and to improve the rate of SA model utilisation.

Having found that software architectural modelling languages are NOT frequently used by many developers (50%), the responses to the next two questions (Q6 and Q7) relating to factors that could be affecting the utilisation of these models either positively or negatively assumed some importance.
In order to assess this, **thirteen factors** that could possibly encourage/discourage the use of modelling languages during SA development were proposed through questions Q6 and Q7. Analysis results are illustrated in Figures 5.6, 5.7, and Table 5.8.

![Figure 5.6: The main factors Encouraging the utilisation of modelling techniques to describe SA.](image-url)

A majority of the respondents believe that demonstrating SA concepts and features using models is easier, and is the *main factor that encourages* the utilisation of models. A further 30% attributed the utilisation of models as making designer and programmer jobs much easier. While, 24% of the respondents, select the statement “it makes the evaluation of stakeholders’ requirements for quality attributes possible in the early stages of the development life cycle”. Other encouraging factors identified are illustrated in Figure 5.6.

The implication of these findings are discussed in more detail in Section 5.3.3.3 since they form part of the main goals of this study.

The summary of responses to 5 possible *factors that discourage the utilisation* of modelling techniques during SA description are illustrated in Figure 5.7.

Out of the five factors listed, 38% of all the respondents believe that the main factor discouraging the utilisation of modelling techniques is “hard to integrate these models with other artefacts (e.g. design models), so they become stand-alone models, which to a degree is *NOT* that useful during the development of software/systems”. To illustrate, the inherent difficulty (and even lack of time) to incorporate changes in the later parts of the development cycle need to be reflected in earlier documentation. Con-
Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system.

Lack of standardisation between existing architecture modelling technique, notations, and semantics.

Current architecture description languages (including modelling languages) are still immature.

Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money.

Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance).

Figure 5.7: The main factors Discouraging the utilisation of modelling techniques to describe SA.

nectivity between architecture and design elements become increasingly fragile (or even irrelevant) when high levels of structural and functional change in the code are not updated in the design etc.

One of the other factors that hinders the use of software models is the difficulty in evaluating the models against the stakeholder’s QAs requirements. There is, therefore, a need for selection of an optimum modelling language that can enable the models to be useful to all stakeholders as well as enabling qualitative and quantitative evaluation. As a result, it will improve current evaluation techniques. Many forms of language can be applied for describing SAs, including natural language, semi-formal language, and formal language, as explained in Chapter 2, which can be used jointly in any combinations to describe SA, or mixed in a scientific approach that could produce a better common language.

The 8th question of the survey, sought to determine the best languages to describe software/system architecture, so as to be more useful to all stakeholders and to support qualitative and quantitative evaluations. The analyses are shown in Table 5.5 and Figure E.1.

More than half of the respondents (52.2%) believe that the best language to use for software description is a combination of natural and semi-formal languages. A Chi-Square test performed on the results confirms that the frequencies in each category are NOT uniformly distributed. The difference is thus significant.

According to the results, it is observed that the best architecture/modelling description language depends on its formality level and usability factors. Natural languages, such as texts, are easy to draft and use during the development of SA models. Furthermore, natural languages are preferred more due to their familiarity to stakeholders, rather than the SA modelling and description languages, with which they are unfamiliar.

However, “Natural language” could cause a conflict between stakeholders, due to different interpretation of the written text among them. Furthermore, it may NOT contain any visual aspects or precise semantics, making the designers job arduous to follow the architect’s lead.
“Formal languages”, on the other hand, have a systematic structured, which could be effectively used to track (software) models within later developed artefacts such as code. Also, they incorporate visual aspects that make it easier for the developer to model complex systems while visualising each step of their architecture clearly. However, similar to other computer languages, these languages require knowledge and experience before using them effectively.

A trade-off between a language’s simplicity and formality must be acknowledged, in order to select the best language to use for SA development. Thus, it makes sense regarding the respondents’ choice between semi-formal language and natural language as their best modelling language to use for SA description, especially, after the improvements over the last decade in the field of SAs description languages and supporting tools. A combination of the two can provide users with good features that enable them to trace requirements through software architecture/design, but they still have their disadvantages, which has been discussed in Chapter 2.

Since this study is aimed at contributing towards the development of a systematic architecture evaluation domain, it was fundamental to determine the suitability of modern architectural frameworks within system architecture description. There are many modern architectural standards and frameworks including (e.g. ISO/IEC 42010, DoDAF, RUP/4+1), among others. According to the results illustrated in Table 5.6 and the box plot in Figure E.2, there is a significant agreement with the statements being made. However, further analysis using a one-sample T-test indicates that the median value is larger than the neutral value (3). This indicates a high tendency of neutrality of opinion from respondents regarding a statement.

Table 5.6: Descriptive statistics for Likert scaled items along with one-sample t-test results for testing assumed mean of 3 (Neutral value), (Q9, Q10, Q15-Q23).

<table>
<thead>
<tr>
<th>Item</th>
<th>N</th>
<th>mean</th>
<th>SD</th>
<th>t-statistics</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture.</td>
<td>45</td>
<td>3.47</td>
<td>0.76</td>
<td>4.14</td>
<td>.000</td>
</tr>
<tr>
<td>Usage of software style/pattern concepts and models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</td>
<td>46</td>
<td>3.00</td>
<td>0.84</td>
<td>0.00</td>
<td>.500</td>
</tr>
<tr>
<td>“Architecture is design, but NOT all design is Architecture”.</td>
<td>45</td>
<td>3.84</td>
<td>0.85</td>
<td>6.65</td>
<td>.000</td>
</tr>
<tr>
<td>There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers.</td>
<td>45</td>
<td>3.80</td>
<td>0.59</td>
<td>9.13</td>
<td>.000</td>
</tr>
<tr>
<td>Most of the existing software architecture evaluation methods, produce qualitative results.</td>
<td>44</td>
<td>3.32</td>
<td>0.86</td>
<td>2.46</td>
<td>.009</td>
</tr>
<tr>
<td>It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures.</td>
<td>43</td>
<td>3.70</td>
<td>0.80</td>
<td>5.70</td>
<td>.000</td>
</tr>
</tbody>
</table>
It seems that such modern frameworks are more organised and can enable developers to save time while coming up with more reliable and reusable models. However, the complexity of such modern frameworks cannot be ignored either, which could be the possible reason why most of the respondents tended to be neutral about the statement. In order to encourage the embrace of such modern architectural frameworks, it may be necessary to promote their incorporation into academic syllabuses to enable many developers to understand how they work, and consequently incorporate them into their work.

Apart from the use of modern architectural frameworks, the use of software patterns is also likely to increase the usage of modelling description languages. However, as noticed from Chapter 4 analyses, software patterns have NOT been strongly embraced during the development of software architecture/design. Thus, this study sought to determine the level of agreement of respondents to the second statement in Table 5.6. The respondents remained neutral about this statement. Neutral responses are hard to interpret. However, in this case the first two statements above support previous Chapters 2 and 3 discussion. However, the complexity of using these technologies can be a hindrance factor that could prevent developers from utilising them within software development.

Further analysis confirms that there seems NOT to be many architectural tactics or metrics in existence, with which to carry out evaluations of architectural models. Only 15% of respondents agreed that they were aware of system/software architectural tactics or metrics that have been, or are being, used for evaluating architecture models (e.g. detecting attacks for security).

The results are shown in Table 5.5, and Figure 5.8. The result was surprising, because about 85% of the respondents lacked this awareness. Further research is recommended to determine the cause of such unawareness. This result raises three points for discussion:

• Whether it is because such metrics do NOT exist, which is NOT correct because they do, such as the work by Kan [2003], Grady [1992], Jaquith [2007], Bass et al. [2013], and AV Sriharsha [2015].
• Is it simply the lack of awareness by developers? This could be affected by different factors, such as current curriculum ignorance of metrics/tactics. Alas, no references have been given by respondents in the comment field.
• Or are they NOT mature enough to be applied in the industry, and they lack tool support? This is partially true, as found in this study’s preliminary research’ findings.
In order to achieve software of high quality, the architect needs a guide to select the best architectural patterns and design tactics. A method is therefore required to analyse and quantify the interactions that exist between QAs, patterns, and tactics. This method is called quantitative evaluation, Kassab et al. [2012]. Hence, the survey sought to determine the extent to which quantitative evaluation methods are known and used among developers.

Respondents were asked if they knew or used any architectural evaluation method that could produce quantitative measures surrounding architecture characteristics. Only 10.9% of the respondents either knew or used such evaluation methods as shown by the results illustrated in Table 5.5, and Figure E.3. It seems clear that software quantitative evaluation methods have NOT been embraced by developers compared to qualitative evaluation methods.

One of the factors that could support SA quantitative evaluation, as identified by 51% of the respondents, is the availability of tools that are required to describe and evaluate SA. A second factor, is the use of a standard language and standard architectural framework for describing SA. The formality level of the SA languages can also influence quantitative evaluation methods. Lastly, the process can also be improved, depending on the documentation mechanism used during SA description. The full results of the analysis are listed in Figure 5.9 and Table 5.8.

Since evaluation is a critical process for eliminating any vulnerability in the system or software, the right tools are needed to effect the evaluation. In addition to the proper tools for evaluation, it is important that the language and framework used for evaluation are standardised. This makes the process
easier, consequently, attracting many developers and architects to use tools or improve them to contribute to SAE. Therefore, the findings above make sense and put more emphasises on the goal of this research, which is to standardise the process of SAE, and to come up with a systematic framework that can be used in the future.

As determined earlier, most of the respondents have preferred moderate levels of formality, during SA description, to backup their quantitative evaluation in later stages. It is, thus, NOT surprising that the correct choice of this formality level, is considered one of the major factors that support SAE.

Furthermore, the study also sought to determine the extent to which the same factors could hinder the process of SAE, as shown in Table 5.8. This part of the survey can provide useful insight into the aspects that currently need to be improved, and to have immediate attention given to encourage quantitative evaluation of SAs.

Three major factors that hinder SA quantitative evaluation include:

1. The formality level of the SA description language.
2. The language used to describe the SA.
3. Tool availability for SA description and evaluation.

The findings form an interesting trend, compared to the factors that are likely to support quantitative evaluation as described earlier. It is noted that, while 27.9% of all the respondents agreed the level of language formality that was used for SA description is likely to support quantitative evaluation, half of the respondents identify the same factor as an obstruction for SA quantitative evaluation. This is an indication that there is a potential to improve quantitative evaluation utilisation, by selecting the proper level of formality. Therefore, focusing research future on the level of the SA formality level, and its effect on SA quantitative evaluation, is recommended.

In a similar way, while 16.3% of the respondents believe that the language used in describing SA is likely to support the process of quantitative evaluation, another 40.5% of the respondents identified it as one of the critical factors that hinder the process for evaluating SAs quantitatively, as illustrated in Table 5.8.

Thus, focus should also be given to the availability of the right tools to describe and evaluate SA. While it was identified as the most important factor that is likely to support quantitative evaluation of SA, still there is a substantial number (35.7% of all the respondents) who believe that it is one of the factors hindering the same process. The other factors that should NOT be ignored include the use of standard language and architectural framework for describing SA, as well as the documentation mechanism used in the process of SA descriptions.

More significant analysis for questions Q15-Q19 shows an agreement to all the five statements as represented in Figure 5.10, where the mean of responses to each statement being on the right of the neutral line. However, the strength of the agreement to each statement is variable.

Therefore, it could be concluded that all the five factors discussed above have a significant influence on quantitative evaluation methods and their usage. In order to improve them and their utilisation, it is paramount to give each factor attention, to increase their support towards quantitative evaluation methods, and to decrease their negative impact.

Finally, this study continued by analysing the effectiveness of current technologies including: (methods, tools, languages, and models) within the SAE, and its automation process. This was done through
5.3. SURVEY METHODOLOGY AND PROCESS

“Architecture is design, but NOT all design is Architecture”.

There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers.

Most of the existing software architecture evaluation methods produce qualitative results.

It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures.

Reliable tools are important for developing or evaluating software/system architectures.

<table>
<thead>
<tr>
<th>Error bars: 95%CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>2.0</td>
</tr>
<tr>
<td>3.0</td>
</tr>
<tr>
<td>4.0</td>
</tr>
<tr>
<td>3.8</td>
</tr>
<tr>
<td>3.8</td>
</tr>
<tr>
<td>3.3</td>
</tr>
<tr>
<td>3.7</td>
</tr>
<tr>
<td>4.3</td>
</tr>
</tbody>
</table>

Figure 5.10: Means for the statements in (Q15-Q19), with error bar and assumed mean line.

analysis of the last four statements (in questions Q20-Q23 of the survey). The analysis of respondent’s agreements are shown in Table 5.6, and Figure 5.11.

There was an agreement by more than half of the respondents to the statement given by Q20. Only 7% disputed the statement, while 38% took a neutral stand. Since, the distribution of the responses was negatively skewed **BUT** mesokurtic, there was, therefore, an agreement to the statement with a mean of 3.5, and a Standard Deviation (SD) of 0.7. The results imply that more aggressive effort is required to develop a standard evaluation framework that should be supported by better tools.

Furthermore, 43% either agree or strongly agree with the statement in Q21. The greater number of the respondents, (48%) took a neutral stand regarding the statement, while only 9% dispute the statement. There is **significant** agreement with the statement, with a mean of 3.4 and a SD = 0.8, which indicates that this research is a challenging task (based on its objectives) and a valuable contribution to the software community.

Also, there was **significant** agreement by respondents to the statement made in Q22, with more than half of the respondents agreeing to the statement. There was a mean agreement of 3.4 with SD = 0.7, which was statistically **significant** at a 1% interval. Hence, while the use of different modelling languages for architectural description may achieve better QAs such as security, it may make the software evaluation processes very complex. Alternatively, restricting SA description to a specific modelling language elevates the consistency level, consequently, making SAE easier.

Lastly, about a third of the respondents either disagreed or strongly disagreed with the statement shown in Q23. A further 42% remained neutral about the statement, with only 26% either agreeing or strongly agreeing. **According to this research investigation, which has been conducted progressively since 2010 until April-2018, there is no approach with supporting tools which can perform such a function yet.** Nevertheless, current technologies are making tremendous progress bringing them close to achieving the task described in the statement above.

Generally, agreement to the first three statements above (Q20-Q22), and disagreement to the last statement (Q23), indicates the shortage of appropriate tools for describing and evaluating SA. Thus,
Current technology lacks reliable software architecture evaluation tools.

Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task.

Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier.

Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes.

Figure 5.11: Means for the statements in (Q20 to Q23), with error bar and assumed mean line. More research is recommended to determine how current technology can be used to develop such tools, in order to improve the process of SA description and evaluation.

5.3.3.1.2 Two Dimensional Analysis Results

The fundamental findings of the survey were revealed through a one-dimensional analysis, as reported in Section 5.3.3.1.1. However, in addition to the individual analysis findings, there could be additional underlying 'correlations' between the answers to different pairs of questions. Based on this, several patterns and trends in the field of SA and SAE could be revealed by analysing the answers to questions in pairs to show any correlation that could exist between each pair.

The purpose of a two-dimensional analysis was to determine:

- Any underlying relationships between the results of various pairs of questions. These trends can aid in developing appropriate recommendations for improving aspects of the software architecture domain.
- To uncover and/or resolve any factors or challenges that could obstruct the development of a concrete SAE framework.

In the quest to discover the relationship that exists between the response/agreement to the statement in Q9 and the general field of expertise of the respondents, a t-test was conducted to compare the equality of the means of the two items. A significant relationship was found between the two items, with the higher group mean for the item corresponding to the respondents whose general field of expertise was from the “Architecture” field. Thus, the highest agreement rate to the statement comes from architects. These results are illustrated in Table E.3.

Similarly, architects agree more to the statement “Architecture is design, BUT NOT all design is architecture”. These results are shown in Table E.6.

In the process of two-dimensional analyses, some results have non significant relationship. Thus, analysis of the answers to questions that showed significant relationships, and their implications is discussed in this section.
5.3. SURVEY METHODOLOGY AND PROCESS

On the other hand, most respondents whose field of expertise is “Project Management” tend to disagree with the statement in Q17 with a mean difference = -0.67, t=-2.35, p-value < .05, as shown in Table E.4.

For the same statement (in Q17), designers showed the second highest level of disagreement, with a mean difference = -0.54, t=-2.1, p-value < .05. Also, they showed a higher level of agreement on the statement in Q19, “Reliable tools are important for developing/or evaluating software/system architecture”, with a mean difference = 0.37, t=2.04, p-value < .05, as illustrated in Table E.5.

Further analysis showed that respondents whose software expertise is in coding (programmers), have the higher level of disagreement to some statements compared to other experts as shown in Table E.7.

1. They (programmers) disagreed more with the statement, “Architecture is design, BUT NOT all design is architecture” – Q15.
2. They also showed a higher level of disagreement with the statement, “it is worthwhile to undertake an effort to develop a quantitative methodology for evaluating software architectures”.
3. Finally, they tend to disagree more than other experts with the statement in Q23.

Similarly, respondents whose field of expertise is “Testing” disagree with the statements in Q15 and Q23, as illustrated in Table E.8.

The results above highlight a difference in experts’ opinions regarding SA, depending upon their area of expertise. The results make sense, because different professions have different needs regarding the use of SA models. This emphasises the importance of having a wide range of specialities (within the software domain) included in this research, which gives more credibility to the results and their interpretation.

As stated earlier in this study, software architecture and design is a high abstraction concept compared to programming. It is therefore expected that most programmers would disagree with some statements or suggestions that are proposed by the researcher, which could be an indication of their lack of architectural knowledge.

A one way ANOVA analysis was also conducted in an effort to determine whether there is any significant relationship between the sectors in which respondents worked and the postulated statements by researchers, as tabulated in Table E.9.

For the statement presented in Q10, a significant variation was noticed with different sectors of the economy. An Least Significant Difference (LSD) test was therefore conducted to provide a further analysis, such as, which sectors disagreed or agreed more with the statement. The analysis is shown in Table E.10.

The results show that academics tend to agree more with the statement. They also make sense, since software developers who gain most of their experience in the academic sector are probably more research oriented compared to the rest. Therefore, they can detect improvements and changes concerning the SA domain.

Furthermore, the analysis sought to determine the relationship between respondents’ awareness regarding architectural tactics or metrics and their work sectors. According to the results shown in Table E.11, there is a significant relationship between the respondents’ sectors and awareness. Hence, most of the respondents who gained most of their experience in industry seem to be aware of such tactics, whereas the greatest numbers of respondents from academic seem to lack awareness about architecture.
tactics and metrics. This might be due to the fact that more effort has been applied in some sectors to promote awareness about the evaluation tactics or metrics, than others. As a result, this study serves to improve strategies to promote such awareness across all sectors and to eliminate obstacles through its contribution.

To gather possible factors that could discourage the utilisation of modelling techniques during SA description, the study shows that there is a significant relationship between the number of years of experience that respondents had in software development and those factors. According to the results in Table 5.7 and E.45, most of the factors seem to be identified by those who had a shorter period of experience in software development. This makes sense, because as one starts using SA modelling techniques, they are more likely to encounter more challenges, because they are unfamiliar with the (modelling techniques) domain. However, with longer experience, developers are more likely to be familiar with most of the SA concepts. Thus, they are able to eliminate most of the factors that could discourage them from utilising SA modelling techniques. The same relationship was also noted with the main factors that could encourage the utilisation of modelling techniques to describe software architecture.

Table 5.7: Cross tabulation between respondents’ experiences and the main factors that discourage the utilisation of modelling techniques to describe SA, (Q3 with Q7).

| What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture? You may select up to two options. | How many years’ experience do you have in total in the software/systems development field? |
|---|---|---|---|---|---|
| | 5–10 (years) | 10–15 (years) | 15–20 (years) | 20–25 (years) | Over 25 (years) |
| Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system | 8 | 3 | 3 | 4 | 1 |
| Lack of standardisation between existing architecture modelling technique, notations, and semantics | 7 | 1 | 2 | 4 | 3 |
| Current architecture description languages (including modelling languages) are still immature | 0 | 4 | 1 | 2 | 1 |
| Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money | 7 | 4 | 0 | 1 | 0 |
| Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance) | 8 | 4 | 0 | 0 | 1 |

A significant relationship was also noted between the agreement and disagreement with the statement in Q17, “Most of the existing software architecture evaluation methods, produce qualititative results” and
respondents’ experience. The ANOVA analysis in Table E.12 reveals that respondents who have (5–10) years of experience tend to agree more with the statement, compared to those with 25 years of experience. At the same time, the results also reveal that respondents with (10–15) years of experience have a greater tendency to agree with the statement. These results are an indication that the opinions of the respondents regarding the existing software evaluation methods depend on their experience. The reason why respondents with less experience tend to agree more with the statement, is probably because the level of required work, goes hand-in-hand with the level of their knowledge. The more experienced one becomes, the more requirements will be assigned to him/her. Thus, respondents with over 25 years of experience could be having much more expectations beyond existing software evaluation methods. However, based on my experience and through this research investigation, the SA qualitative evaluation methods are more concrete and have been/are being applied in industry, compared to the quantitative methods that are still immature and need more effort in order to be ready for deployment in the real world.

As noted earlier, the respondents’ years of experience may have a direct bearing upon their awareness of software/system architecture description/modelling languages. On the basis of such relationships, the opinion of respondents regarding the automation of the evaluation process could also be affected. I would expect that the most credible opinions would come from those who are aware of SA description/modelling languages. According to the results in Table E.14, a higher rate of agreement with the statement in Q21, “Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task”, was from respondents who are aware of the existence of SA description/modelling languages, compared to those who are unaware.

This leads to the conclusion, that the agreement with the statement in Q21, is mostly right as discussed in Chapter 2. This was also the case, regarding the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”– Q20.

More analysis using one way ANOVA test, between Q5 and Q16, shows the difference among group means for the statement in Q16, corresponding to the categories under how often respondents used models. Tables E.15 and E.16 shows the results of the test. A significant result was realised with, $F = 2.7$, $p$-value < .05. The multiple comparison test results using LSD method reveal that the group of respondents who nearly always used models for SA description had a higher rate of agreement, compared to those who never used them, infrequently used them, and reasonably/frequently used them. This is probably because the respondents who used models more frequently could discern the statement more clearly. Also, it could also be due to the possibility that the more developers become familiar with the models during their use, the clearer the distinction between architecture and design becomes. Consequently, they can differentiate between architectural models and design models. I agree with the respondents that the current literature still needs to be clearly documented and simplified in an organised manner to eliminate any confusion that may be hindering software developers from utilising models during SA description.

Moreover, in relation to the models’ frequent use for SA description, the group means for the statement in Q18, “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures”, exhibited a significant difference, $F = 3.0$, p-value .05, as illustrated in Tables E.17 and E.18. As a result, there is a distinct relationship between mean responses to this statement and their frequent use of models. This implies that the developers’ agreement to this statement varies based on their familiarity with models. The more developers use models, the more they see the importance of developing a quantitative methodology as proposed in the statement in Q18, thus elimi-
nating discouraging factors considered as obstacles to the use of models. Hence, the agreement to the statement could increase the usability factor of models among developers, and their knowledge, as well as supporting the value of the proposed solution suggested within the statement at Q18.

Another interesting result in relation to how often respondents used SA models with the statement in Q22, “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier”, as shown in Table E.20. The means are significantly unequal for the respondents who “Infrequently (<10%)” used models to those who “Regularly (>50% and <80%)” used them, with Mean difference = 0.9, p-value < .05.

The differences exhibited are an indicator of the effect of length of experience in software/system development. The more experience a developer has, the more chance that he/she would use models frequently for describing SA. That is why in this study I chose a minimum 5 years of experience to participate in the survey, in order to ensure that the survey answers and analysis are reliable.

More significant relationships are noted between the respondents who agree that the use of models “makes the designers/programmers job much easier” and those who think that “Reliable tools are important for developing/or evaluating software/system architectures”, as shown in Table E.21. This relationship is expected, since reliable tools should create consistent models that would help designers/programmers do their work much easier.

As a result, NOT only does the designing work become easier, BUT a systematic model is also developed to support easier evaluation processes. Therefore, this study emphasises an important choice for the proper tools for SA development.

The last significant correlation in this section was noticed between responses to the statements in Q10 and Q17, (“Usage of software style/pattern concepts & models during architecture development increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture evaluation” and “Most of the existing software architecture evaluation methods, produce qualitative results”), as presented in Table E.24 and E.23. A Chi-square was used to test the independence between their categorical variables. The test is significant, $\chi^2 = 30.52$, p-value < .05. This confirms that there is an association or dependence between the categorical variables, which could be interpreted in different ways.

However, one main objective of patterns is to simplify the solutions. So, SPs might increase model utilisation within architectural levels, and consequently increase the simplicity of the evaluation process, (if and only if ‘SPs are developed and documented well’),

Taking that into account, the qualitative evaluation methods might NOT have been affected directly by SPs due to their nature.

While quantitative methods could be impacted by patterns (positively or negatively), if developers include them within their architectural models, they will also be affected by the patterns’ structures, integration mechanism, and maybe their documentation. Both factors, patterns integration and documentation mechanism, are important, as described earlier in Chapter 3.

5.3.3.2 Inappropriateness of Three and Four Dimensional Analyses

As in Chapter 4, three and four dimensional analysis of the responses proved to be non-significant. Analysis of both dimensions for some of the questions within this survey were carried out (see appendix section E.4), but the results were inappropriate due to the same reasons explained in Section (4.3.6.3) for
the first survey.

5.3.3.3 Focused Analysis on Q6, Q7, Q13 and Q14

More emphasis on the relationship between the four questions (6, 7, 13, and 14) and other statements within the survey is preferable, because they comprise the important factors (18 in total) that are strongly related to the research objectives. Since, they could have an effect on SA (modelling techniques and quantitative evaluation). These analyses are discussed more in this section, in order to contribute to the overall research objectives, to help to identify more constraints concerning utilising the SA modelling techniques and quantitative evaluation, and to suggest possible solutions that could aid in developing a systematic framework, which can be used to ease the process of SAE in the future. The frequency distributions for the 4 questions are illustrated in Table 5.8.

5.3.3.3.1 Individual Analysis

One-dimensional analysis for the four questions is discussed in this section. The distribution of the responses for the four questions in Table 5.8, are ordered as below:

1. Factors that ENCOURAGE the utilisation of modelling techniques (Q6).
2. Factors that DISCOURAGE the utilisation of modelling techniques (Q7).
3. SUPPORTIVE factors for quantitative evaluation (Q13).
4. HINDERING factors for quantitative evaluation (Q14).

The responses indicate that the discouraging factors outweigh the encouraging factors; hence, the developers are discouraged more from using models to describe SA. This also might be an indication to do more work within this domain, in order to make the models development process more effective and easy to use.

Table 5.8: Frequency distribution of the multiple response questions/items, (Q6, Q7, Q13, Q14)

<table>
<thead>
<tr>
<th>Question/Item</th>
<th>Frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The main factors that ENCOURAGE the utilisation of modelling techniques to describe software/system architecture (N=46)</td>
<td></td>
</tr>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
<td>30 (65.2)</td>
</tr>
<tr>
<td>Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily</td>
<td>8 (17.4)</td>
</tr>
<tr>
<td>It makes the designers/programmers job much easier</td>
<td>14 (30.4)</td>
</tr>
<tr>
<td>Makes the evaluation of stakeholders requirements for quality attributes</td>
<td>11 (23.9)</td>
</tr>
<tr>
<td>Possible in the early stages of the development lifecycle</td>
<td></td>
</tr>
<tr>
<td>Reliable modelling tools for describing the architecture exist, which</td>
<td>4 (8.7)</td>
</tr>
<tr>
<td>makes the usability factor much easier</td>
<td></td>
</tr>
<tr>
<td>The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible</td>
<td>2 (4.3)</td>
</tr>
<tr>
<td>Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, UML, UML )</td>
<td>6 (13.0)</td>
</tr>
<tr>
<td>Teaching of the architecture modelling languages in the academic sectors</td>
<td>8 (17.4)</td>
</tr>
</tbody>
</table>
The main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture (N=45)

<table>
<thead>
<tr>
<th>Factor</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system</td>
<td>19 (42.2)</td>
</tr>
<tr>
<td>Lack of standardisation between existing architecture modelling technique, notations, and semantics</td>
<td>17 (37.8)</td>
</tr>
<tr>
<td>Current architecture description languages (including modelling languages) are still immature</td>
<td>8 (17.8)</td>
</tr>
<tr>
<td>Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money</td>
<td>12 (26.7)</td>
</tr>
<tr>
<td>Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)</td>
<td>13 (28.9)</td>
</tr>
</tbody>
</table>

Important factors that could SUPPORT quantitative evaluation for any SA (N=43)

<table>
<thead>
<tr>
<th>Factor</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>7 (16.3)</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>12 (27.9)</td>
</tr>
<tr>
<td>Using standard language and architecture framework for describing SA</td>
<td>20 (46.5)</td>
</tr>
<tr>
<td>Tools availability for describing and evaluating SA</td>
<td>22 (51.2)</td>
</tr>
<tr>
<td>Documenting mechanism used during SA description</td>
<td>9 (20.9)</td>
</tr>
</tbody>
</table>

Important factors that could HINDER quantitative evaluation for any SA (N=43)

<table>
<thead>
<tr>
<th>Factor</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>17 (40.5)</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>21 (50.0)</td>
</tr>
<tr>
<td>Using standard language and architecture framework for describing SA</td>
<td>5 (11.9)</td>
</tr>
<tr>
<td>Tools availability for describing and evaluating SA</td>
<td>15 (35.7)</td>
</tr>
<tr>
<td>Documenting mechanism used during SA description</td>
<td>6 (14.3)</td>
</tr>
</tbody>
</table>

In addition to identifying these factors, recognised efforts have already been made in this field by researchers as mentioned in Chapter 2, in order to produce solutions that could minimise the effect of some hindering factors. Also, this research could render a positive contribution to this domain, having identified the need for further research to uncover more ways for developing, suitable solutions, to overcome hindrances, and for promoting encouraging factors within the SA domain.

One of the interesting findings to be noted from the analyses, is the divided opinions regarding the contribution modelling could make to SA evaluation becoming more effective. This is because the evaluation of SA in general, is a fundamental process of SA development, which is affected by the way SA is described in the first place. Thus, the developers familiarity with the SA formal description languages and modelling techniques, support them to embrace quantitative evaluation, while others who are unfamiliar with them will try to bypass them by choosing different approaches. It is thus important to spread the knowledge regarding SA modelling methods and their qualitative and/or quantitative objectives.

One of the main factors that could support SA quantitative evaluation is “the availability of tools for describing and evaluating software architecture”. The following factor (in Table 5.8) is the “use of standard language and architecture framework for describing SA”. Other factors follow, as illustrated in the table. A total of 39 (out of 43) respondents identified factors that are related to language used during SA description.
Therefore, in order to influence the use of quantitative evaluation methods, one should be concerned not only with the tool availability, but also it is important to pay close attention to the languages that are used for describing SA, in terms of their formality, standardisation, and nature. Finally, it is also important to improve the documentation of SA description, to increase the visibility of its artefacts.

Further analysis concerning the hindering factors, reveals that the “formality level of SA description” is one main reason that prevents developers from utilising quantitative SAE methods. The other main factor, as identified by respondents, is “the language used for describing the SA”. Two main factors identified by the respondents concern tools and description languages. This seems a valid outcome, based upon the preliminary investigation of this research in Chapter 2, and one that supports some findings, such as, the attempts that have been made to classify the level of SA formality by Rushby [1993b], in order to aid architects to know the suitable level of formality during SA development.

5.3.3.3.2 Significant Results - two dimensional (2D) analysis

Significant relationships exist between the four questions and other statements within the survey, which are described in this section, (by applying two dimensional analysis).

By comparing all the responses to questions that involved a nominal scale, one of the significant relationships discovered was between “how often respondents use models”, and “the main factors that encouraged the utilisation of modelling techniques to describe SA” as illustrated in Table 5.9. A \( \chi^2 \) test conducted on this cross-relationship shows that the result is significant at 1% level (\( \chi^2 = 56.5, \) p-value < .01), which shows a very strong association between the two items. Developers, who used models more than 15% in their work, supported encouraging factors more than those who used models less frequently.

Table 5.9: Pearson Chi-square test results between questions Q5 and Q6.

<table>
<thead>
<tr>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>What are the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture? You may select up to two of the following options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>56.5</td>
</tr>
<tr>
<td>df</td>
<td>32</td>
</tr>
<tr>
<td>Sig.</td>
<td>.005</td>
</tr>
</tbody>
</table>

A similar relationship was also noticed (at 5% level) between answers to both questions shown in Table 5.10 (\( \chi^2 = 33.5, \) p-value < .05). This result indicates that developers using models less than 15% intended to identify their concerns and obstacles by choosing more discouraging factors, compared to those who used models more frequently. The expectation of a direct relationship between the two pairs of items analysed above is predictable.

Hence, model usability is affected by both encouraging and discouraging factors that have been identified by questions (6 and 7). Thus, it is recommended to find solutions for the hindering factors in order to increase and help developers use models more during their work regarding SA description.

Another significant association (\( \chi^2 = 30.44, \) p-value < .07) is between “the main discourage factors to use models” and “the years of experience”. The results of this analysis are shown in Tables 5.7 and E.47.

This association was expected, since developers who have less experience might never use models to describe SA. As a consequence, they may not be able to determine the reasons (or factors) that influence
modelling techniques usability, due to their lack of knowledge and brief experience with modelling, as opposed to those who have more experience and use models more often.

Table 5.10: Pearson Chi-square test results between questions Q5 and Q7.

<table>
<thead>
<tr>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture? You may select up to two options.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chi-square</strong></td>
<td>33.5</td>
</tr>
<tr>
<td><strong>df</strong></td>
<td>20</td>
</tr>
<tr>
<td><strong>Sig.</strong></td>
<td>.029</td>
</tr>
</tbody>
</table>

As a result, in order to improve the process of identification of these factors more precisely, it is necessary to encourage developers to use modelling techniques in SA description more frequently, potentially generating a pool of individuals with longer experience suitable for participation in similar surveys or interviews in the future. The results also reveal, that respondents with over 25 years experience identified less hindrances to their utilisation of models during their work. This could mean, that utilisation of models becomes easier and more efficient as the developers gain more experience using them. Although, the results could be affected by the percentage of respondents with over 25 years experience, which was too low to be significant.

Moreover, to identify the relationship between both statements (Q7 and Q10) shown in Table 5.11; a null hypothesis, $H_0: \mu_1 = \mu_2$ was developed alongside an alternative hypothesis, $H_a: \mu_1 < \mu_2$ where: $\mu_1$ = the population mean of the selected item (the column variables) for the respondents who answered, “YES” to the statement “Hard to integrate...etc”.

$\mu_2$ = the population mean for the respondents who answered “NO” to the same statement.

An independent sample t-test showed a significant results at the 5% level, $t=-1.8$, $p-value<.05$. The lower population mean for those who answered “Yes”, compared to those who answered, with “No”. The results are shown in Table 5.11.

The significance of the test shows that the population mean of statement-1 “Usage of software style/pattern concepts ...etc” is significantly lower for the respondents who replied, “Yes” to statement-2 “Hard to integrate models ...etc”, than those who replied, “No”. Hence, the respondents who agreed more with the first statement, disagreed more with the second. Thus, the respondents who believe that SPs increase model usage and make the evaluation process harder, also believe that models are useful and easy to integrate. One explanation for this result is that there are some SP users dealing with patterns as small models, and that allows them to integrate those patterns with other components of the software. Thus, they see it as easy to integrate models.

However, it’s better if the first statement is divided into two separate parts (statements) to avoid confusion and to provide precise analysis. The first part should be “Usage of software style/pattern concepts and models during architecture development increases the utilisation of modelling description languages”; whereas, the second statement should be “Usage of software style/pattern concepts and models during architecture development decreases the simplicity of the architecture evaluation”. This important separation was identified during the analysis, because maybe people who agreed with the first part of the statement, disagreed with the second part, which forced them to provide inaccurate answers, and vice versa. Such remarks will be taken into account in future research.
Table 5.11: Independent samples t-test results between both statements for Equality of Grouped Population Means corresponding to the categories (Yes or No).

<table>
<thead>
<tr>
<th>Item</th>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>Sig.</td>
<td>t</td>
</tr>
<tr>
<td>Item: Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system. Group: “Yes” and “No”</td>
<td></td>
<td></td>
<td>1.8</td>
</tr>
<tr>
<td>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</td>
<td></td>
<td></td>
<td>1.9</td>
</tr>
</tbody>
</table>
Another significant correlation was noticed between the most important factors that could support quantitative evaluation and the statement illustrated in Table E.27 (Q10 and Q13). The result reveals the contribution of software style/patterns toward the SAE, as well as the SA modelling techniques’ domain. However, based on the investigations in Chapter 3 and Chapter 4, the emphasis on better SP documentation, and integration mechanisms still remains.

Table 5.12: Pearson Chi-square test results for analyses of questions Q10 and Q13.

<table>
<thead>
<tr>
<th>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</th>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>34.36</td>
</tr>
<tr>
<td>df</td>
<td>20</td>
</tr>
<tr>
<td>Sig.</td>
<td>.024</td>
</tr>
</tbody>
</table>

Similarly, the evaluation of the statement that “there is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design...etc” around the null hypothesis, $H_0 : \mu_1 = \mu_2$ and alternative hypothesis, $H_a : \mu_1 \neq \mu_2$ where:

$\mu_1 =$ the population mean of the selected statement (the column variables) for the respondents who answered “Yes” to the item “Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, and performance)”.

$\mu_2 =$ the population mean for the respondents who answered “No”.

The results in Table E.22 show a significantly lower population mean for those who responded, “Yes” to the statement, than those who responded, “No”, $t=1.96$, p-value=.028<.05. Thus, a higher population mean for respondents who believes that there is no vagueness between the architecture abstraction and high level design, also believes it is easy to evaluate SA models. However, the research findings disagree with their opinion, where both the confusion between architecture and high level design and the difficulties of SA evaluation still exist within the domain and current literature.

In a similar way, Table E.26 illustrates a significant relationship at 6% level, between the most important factors that could support quantitative evaluation, and the opinion concerning the statement, “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” $\chi^2(20) = 30.75$, p-value<.06. This relationship is an indication of the effect of the use of current technology on factors that could support quantitative evaluation of SA, as well as the use of modelling in SA descriptions. However, the current technologies (including SA description languages, modelling techniques, and evaluation methods) still need more effort and research to be able to allow us to develop a more concrete quantitative evaluation approach, which could be applied and utilised among the software community.

Further, in order to determine the cross-relationship between the responses with ordinal items in Q15 to Q23 with encouraging factors for modelling methods, null and alternative hypotheses were formed, equating the pairs of items. Interesting relationships were noticed. For example, there was a significant relationship at the 5 % level ($t=2.2$, p-value <.05), pointing out that the population mean of the statement “Reliable tools are important for developing/or evaluating software/system architectures” is significantly higher for the respondents who replied, “Yes” to the statement, “It makes the designers/programmers’ job much easier”. This result is an expected relationship, which shows the developers’ enthusiasm to find an
5.3. SURVEY METHODOLOGY AND PROCESS

effective tool for describing and evaluating SA, in order to make their work easier, which will support
the use of models during SA description, see Table E.21.

Similar relationships were noticed in the same table (E.21), between the population means for the
statement “Architecture is design, BUT NOT all design is Architecture”, which is significantly higher for
the respondents who replied, “Yes” to the statement “Makes the evaluation of stakeholders’ requirements
for quality attributes possible in the early stages of the development life-cycle”. The result was expected,
because most of the developers consider architecture and design as an early phase of the software life-
cycle (with which I agree).

5.3.3.3 Insignificant Results - two dimensional (2D) analysis

Besides the significant results identified in the previous section, this section contains a brief summary of the relationships that did NOT prove
to be significant according to the findings, for the four questions (6, 7, 13, and 14). In most cases, these
non-significant results imply that there is no link or relationship between two variable questions under
consideration. However, it could be useful to comment on some independences between the various
pairs. The lack of significant relationships with these variables may help to draw meaningful insights
from the findings, as well as enabling future researchers to narrow down their focus to only the variables
that matter. Consequently, this will improve the time taken to conduct new survey, and hence, improving
the efficiency in general of such studies.

Two-dimensional analyses reveal that none of the four questions have any significant relationship
with the sectors in which the respondents gained most of their general software development experience.
The two dimensional analysis results proved non-significant as outlined in Sub-sections E.3.2.1, E.3.2.2,
E.3.2.3, and E.3.2.4.

The results were unexpected, because normally the work environments are considered to be totally
different across the sectors. Thus, one of the sectors is expected to show a significant result concerning
some of the tested relations. The non-significant results between the work sectors and the four questions
are an indicator of ambiguity regarding the reasons that cause these results. This ambiguity could be re-
solved by an interview/study in the future. The following non-significant relationships are also considered
subject to the same conclusion.

All the two-dimensional tests between the four questions and the awareness of SA description and
modelling languages in Q4 proved non-significant.

Relating to the respondents’ years of experience, Q6, 13 and 14 yielded non-significant results,
whereas model discouragement factors (Q7) do have significant results, as explained earlier.

The frequency use of models by developers (Q5) gave non-significant results when tested with the
factors that discourage modelling in Q7, which indicated that there are no relationships between both
items. This was also the case for the other remaining questions, except the modelling encouraging factors
in Q6, as explained earlier.

Furthermore, independent sample t-tests were performed to determine the association between the
statements (“Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC
42010, DoDAF, RUP/4+1) …etc” and “Usage of software style/pattern concepts & models …etc”) with
the responses to the main encouraging factors for models methods utilisation (Q6). The relationships for
both statements with all encouraging factors, were non-significant at the level of 5%. It was also recogni-
nised that none of the four questions had any significant relationship with each other. Thus, the factors
were independent and isolated from each other.

### 5.3.4 Related Work

Within the findings from this study, some important aspects concerning SPs, SA, SAE, and modelling techniques have been realised. However, it is appropriate to recognise the efforts that have been made by other researchers, which can support the goal of this study. One such study, is reported by Tang et al. [2006], in which the researchers’ aim was to explore the respondents’ opinions about how they perceive design rationale, and how they use it to document SA.

According to Tang et al. [2006], although many designers and architects are aware of the importance of using design rationale, and documenting their concepts, there are still several factors that are a barrier to their use and to documenting their architecture concept. The researchers recommended further research, in order to come up with a methodology, as well as tools to support and capture the design rationale. This research ‘supports’ their recommendations, and could contribute to the design of such a methodology. By tackling the factors that hinder the use of models in software description, this research makes a positive contribution towards the research done by Tang et al. [2006], which has already formed a firm background for this study.

Most of the factors that were identified in this research, including hindrances or supportive factors concerning the utilisation of modelling in SA description, should be useful for the SA language discipline.

Furthermore, the research done by Malavolta et al. [2013], has already attempted to determine the needs of the software industry in terms of the use of architectural languages. Their research has provided empirical evidence about the perception of developers regarding the strengths, limitations, as well as the needs, of the existing Architecture Description Language (ADL)s. In order to extend their research, it is important to investigate the use of ADLs, and SA modelling methods, which is provided by this study. This will aid in the development of future generation languages that will try to solve some of the identified problems and obstacles.

Lastly, is the survey by Ozkaya [2016], which focussed on the academia and industry domains. His survey consisted of 20 questions, and the goal was to reveal the practitioners’ level of knowledge and experience regarding SA.

The survey disclosed three important findings:

- Practitioners’ knowledge on software architectures is too limited.
- Lack of interest by the participants toward ADLs.
- The SAE is a new concept to participants.

There are some other surveys concerning SA and SAE, each with its own scope, that have tackled some specific aspects of SA, but none of them is similar to this study regarding the goal, scope, and types of question. This study, and the other related work mentioned in this section, emphasise the importance of SA artefacts as a part of software components and that SA analysis is critical to verify software designs for quality properties, and in detecting design errors.

### 5.3.5 Limitations

There are a few limitations concerning this survey. The first concerns the survey scope. The scope of the study is too wide and includes different disciplines within the SA domain, such as, description
5.3. SURVEY METHODOLOGY AND PROCESS

languages, modelling methods, and evaluation, where each discipline is worth a separate survey and study. This broadness has advantages and disadvantages; it could be considered comprehensive, but at the same time not precise enough.

The second notable limitation is the use of the questionnaire method, which has commonly known limitations, such as the degree of truthfulness of the respondents, misinterpretation of questions, and the large amount of data that is generated from the open-ended questions, which takes a long time to process and analyse. This type of investigation (questionnaire) needs to be supported by another investigative approach that could either backup or conflict with the questionnaire’s findings, thereby improving the overall knowledge within the domain.

Two good approaches that could get the researcher closer to the developers and development environment are interview and field study methods, which I have introduced in Section 5.4.

Furthermore, most of the questions included in the questionnaire were close-ended. This may limit the variety of opinions gathered from the respondents.

5.3.5.1 Summary of significant results

This section summarises all significant results within the survey discussed in Chapter 5, in order to facilitate the tractability between the findings and related tables, as illustrated in Table 5.13.

Table 5.13: Summary of the main analysis results with associated tables (for better traceability).

<table>
<thead>
<tr>
<th>Findings</th>
<th>Analysis Table</th>
<th>Statistics (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>More than half of the respondents choose a combination of Natural and</td>
<td></td>
<td>( \chi^2 = 60.20 )</td>
</tr>
<tr>
<td>Semi-formal languages, as the best language to use for software</td>
<td>5.5</td>
<td>(.000)</td>
</tr>
<tr>
<td>architecture description.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>About 85% of the respondents were NOT aware of any system/software</td>
<td></td>
<td>( \chi^2 = 23.20 )</td>
</tr>
<tr>
<td>architectural tactics or metrics that have been or are being used for</td>
<td>5.5</td>
<td>(.000)</td>
</tr>
<tr>
<td>evaluating architecture description models.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developers significantly agreed that developing SA using current</td>
<td>5.6</td>
<td>( t = 4.14 )</td>
</tr>
<tr>
<td>architectural frameworks increases the reliability, standardisation, and</td>
<td></td>
<td>(.000)</td>
</tr>
<tr>
<td>reusability of the resulting architecture.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developers significantly agreed that architecture is design, <em>BUT NOT</em></td>
<td>5.6</td>
<td>( t = 6.65 )</td>
</tr>
<tr>
<td>all design is Architecture.</td>
<td></td>
<td>(.000)</td>
</tr>
<tr>
<td>Developers significantly agreed that there is still vagueness in the</td>
<td>5.6</td>
<td>( t = 9.13 )</td>
</tr>
<tr>
<td>current literature concerning the differences between the architecture</td>
<td></td>
<td>(.000)</td>
</tr>
<tr>
<td>abstraction and high level design, which causes some confusion.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developers significantly agreed that most of the existing SA evaluation</td>
<td>5.6</td>
<td>( t = 2.46 )</td>
</tr>
<tr>
<td>methods, produce qualitative results.</td>
<td></td>
<td>(.009)</td>
</tr>
<tr>
<td>Developers significantly agreed that it’s worthwhile to undertake an</td>
<td>5.6</td>
<td>( t = 5.70 )</td>
</tr>
<tr>
<td>effort to develop a quantitative methodology for evaluating SA.</td>
<td></td>
<td>(.000)</td>
</tr>
<tr>
<td>Developers significantly agreed that reliable tools are important for</td>
<td>5.6</td>
<td>( t = 14.47 )</td>
</tr>
<tr>
<td>developing/or evaluating SA.</td>
<td></td>
<td>(.000)</td>
</tr>
<tr>
<td>197</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Developers significantly agreed that current technology lacks reliable SA evaluation tools.  

Developers significantly agreed that reading SA description models for automated evaluation purposes, is a critical, difficult, and error prone task.  

Developers significantly agreed that restricting the description of architecture to a specific modelling language makes the architecture quantitative evaluation easier.  

Architects had significantly more degree of agreement to the statement “Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture” than others.  

Project Managers had significantly less degree of agreement to the statement “most of the existing SA evaluation methods produce quantitative results” than others.  

Designers had significantly less degree of agreement to the statement “most of the existing SA evaluation methods produce quantitative results” than others.  

Designers had significantly more degree of agreement to the statement “Reliable Tools are important for developing/or evaluating SA” than others.  

Architects had significantly more degree of agreement to the statement “Architecture is design, BUT NOT all design is Architecture” than others.  

Programmers had significantly less degree of agreement to the statement “Architecture is a design, BUT NOT all design is architecture” than others.  

Programmers showed less degree of agreement to the statement “It is worthwhile to undertake an effort to develop a quantitative methodology for evaluating SA” than others.  

Programmers showed less degree of agreement to the statement “Current technology allows us to develop general software evaluation models that assess any SA against any QAs” than others.  

Respondents whose field of expertise is ‘Testing’ tend to have less degree of agreement to the statement “Current technology allows us to develop general software evaluation models that assess any SA against any QAs” compare to other experts.
### 5.3. SURVEY METHODOLOGY AND PROCESS

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.8</th>
<th>t = -1.79 ( .041 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respondents whose field of expertise is ‘Testing’ tend to have less degree of agreement to the statement “Architecture is a design, BUT NOT all design is Architecture” than other experts.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.9 and E.10</th>
<th>F = 3.33 ( .045 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academics tend to have more degree of agreement to the statement “Usage of SPs concepts models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture evaluation” than others.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.12 and E.13</th>
<th>F = 3.76 ( .011 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The respondents who had more than 25 years of experiences, had significantly less mean agreement to the statement “most of the existing SA evaluation methods, produce qualitative results” than the respondents who have less experience.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.14</th>
<th>t = 1.90 ( .041 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The mean agreement to the statement “Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task” was higher from respondents who had awareness about the existence of any SA description / Modelling languages as compared to those who lacked such awareness.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.14</th>
<th>t = -2.1 ( .025 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The mean agreement to the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” was lower from respondents who had awareness about the existence of any SA description / Modelling languages as compared to those who lacked such awareness.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.15 and E.16</th>
<th>F = 2.74 ( .042 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The group of respondents who nearly always used SA description models had higher mean agreement to the statement “there is still vagueness in the current literature concerning, etc.” compared to those who never used models at all, those who used them infrequently, or reasonably frequently.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.17 and E.18</th>
<th>F = 2.93 ( .033 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The respondents who nearly always used models to describe SA during their work have higher mean agreement to the factor “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating SA”, compared to the respondents who never used, infrequently used, reasonably frequently used, or regularly used models.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.19 and E.20</th>
<th>F = 2.99 ( .030 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The respondents who infrequently used or reasonably frequently used models to describe SA during their work, have higher mean agreement to the statement “Restricting the description of architecture to a specific modelling language, etc.”, compared to the respondents who regularly used models.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statement</th>
<th>E.41 and E.42</th>
<th>$\chi^2 = 5.43$ ( .066 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>There was an association between the sectors in which respondents gained most of their software development experience and whether they are aware of any system/architectural tactics or metrics that have been or are being used for evaluating architecture description models.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
There was a significant association between the years of experience, and how the developers used models to describe software/system architecture during their work.  
\[ \chi^2 = 26.18 \quad (0.052) \]

<table>
<thead>
<tr>
<th>E.44 and E.43</th>
<th>( \chi^2 = 30.44 ) ( (0.063) )</th>
</tr>
</thead>
</table>

There was a significant association between the modelling discourage factors and the years of experience.

\[ \chi^2 = 33.54 \quad (0.029) \]

<table>
<thead>
<tr>
<th>E.49</th>
<th>( \chi^2 = 56.46 ) ( (0.005) )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>E.46</th>
<th>( t = 2.00 ) ( (0.030) )</th>
</tr>
</thead>
</table>

The mean of the item “Usage of software style/pattern concepts models during architecture development, increases the utilisation of modelling, etc.” is significantly higher for the respondents who say YES to the statement “It makes the designers/programmers job much easier”, compared to who says NO.

<table>
<thead>
<tr>
<th>E.49</th>
<th>( \chi^2 = 56.46 ) ( (0.005) )</th>
</tr>
</thead>
</table>

The main factors that encourage the utilisation of modelling techniques to describe software/system architecture.

There is a significant association between how often respondents use models to describe SA during their work, with the responses to the item “the main factors that discourage the utilisation of modelling techniques to describe software/system architecture”.

<table>
<thead>
<tr>
<th>E.49</th>
<th>( \chi^2 = 33.54 ) ( (0.029) )</th>
</tr>
</thead>
</table>

There is a relationship between how often respondents use models and the responses to the statement “the main factors that discourage the utilisation of modelling techniques to describe SA”.

5.4 Field Study Analysis

5.4.1 Introduction

Hardware and software architecture greatly affect the process of system development, because they act as a foundation of their development. Hence, architects generally qualify as leaders of a majority of system development groups. Architecture remains one of the leading professions within the Information and Communication Technology (ICT) infrastructure, and technology realm.

As a continuous effort to the survey reported in Section 5.3.3, this section describes the findings that are collected from a field study that was performed at a military organisation, in order to evaluate developers’ knowledge regarding SA, and to investigate some architectural aspects, such as SA (description, modelling, utilisation, evaluation, and documentation), within the government and industry sectors, and also to try to gain and/or share knowledge from/with them.

Taking that into the account, knowledge transfer is probably harder than product and service transfer. Transferring new technologies to new areas or environments requires more than simply understanding how a product or method works. There are several different factors that could be involved in the transferring process, such as participants’ background knowledge, time, cost, proper environment, etc..

5.4.2 Objective of the Field Study

Based on Tellis [1997] and references therein pertaining to (Yin, 1994) and (Feagin, Orum, & Sjoberg, 1991), the case study researcher should act as a senior investigator during the field study to obtain the necessary information needed, and as per Yin [2013] providing guidelines and reporting examples.
5.4. FIELD STUDY ANALYSIS

During this study, the researcher was given full access to data, developers, managers, and labs, thus providing enough resources to be able to obtain credible judgements concerning current SA development procedure and documentation within the targeted organisation.

The main objective of this study was to:

1. Identify and observe the development, utilisation, and evaluation of the architecture artefacts in a real world environment. This work may agree/disagree with the research argument reported in the previous chapters surrounding SA.
2. Gather information regarding SA, by interviewing developers who are/were involved in the development of past and current system projects, and by examining old/current SA processes, and documentation.
3. Provide recommendations to the investigated organisation and software community, based on the findings obtained from the aforementioned two points.

5.4.3 Study Process and Methods

This field study was conducted in two phases, each phase includes activities as illustrated in Figure 5.12.

1. Preliminary Phase (two weeks visit), which covered three different organisations. The goal of this phase is to select the environment best suited to the research topic, so as to be able to perform the final phase.
2. Final Phase (two months visit), involved the selected candidate/organisation from the Preliminary Phase. The goal of this phase is to achieve the study objectives.

Figure 5.12: The Field study phases.

5.4.3.1 Organisation Location and Selection Criteria

In order to select an appropriate organisation for this study, the selection criteria were:
• The organisation should be large enough in size, \textit{(No less than 500 employees)}.
• The organisation should have past and current IT projects.
• Project documentation should be available and accessible by the researcher.
• SA modelling techniques and evaluation should be employed by the organisation.

The above criteria were applied in order to ensure that the study environment had enough information, developers, and documents. The projects, documentation, developers, and SA modelling methods were largely dependent on the size of the organisation. Thus, three candidate organisations with no less than 1,000 employees were selected for the Preliminary Phase. Also, their project costs and sizes varied from small-embedded systems or applications, up to large-scale systems. The study’s geographic location was in Saudi Arabia.

5.4.3.2 Communication Procedure

Communication with the three candidate organisations was official and used the following methods:

• Official letter through Saudi Arabian Cultural Mission (SACM) in Australia.
• Phone.
• Email.
• In Person.

Due to the sensitivity of the work of the three organisations, the researcher conducted all of the four types of communication aforementioned in 2012 \(^2\).

5.4.4 Preliminary phase

This section briefly discusses the preparation steps that were carried out, in order to be able to conduct the case study.

In order to explore the environment suitability concerning this study context, \textit{the researcher’s main requirements to conduct this phase were}:

• Approval from the organisation for a preliminary visit to the sites before conducting the field study.
• Assigning (a contact) official person who could facilitate the study.
• Interview and communicate with any developers or stakeholders of the targeted systems under consideration.
• Gain full access to any restricted areas or confidential documents.

The three organisations approved the first three points above, and partial access to the last requirement was granted, which was to both parties (the hosting organisation and the researcher).

\textbf{Organisation sectors:} The three organisations were from the military sector (government); however, all of them have ongoing projects, civilian employees, local contractors, and international contractors. Also, their international contractors are companies, such as Raytheon, Lockheed Martin, and Oracle.

\(^2\)Official communications with targeted organisations were performed one year before conducting the on-site study, in order to be able to receive their approval within this study’s planned time-frame, which was achieved.
The organisations are referred to here as Site 1, Site 2, and Site 3 for confidentiality purposes. Developers within the three organisations were interviewed by the researcher, as illustrated in Table 5.14.

During this process, a prepared check-list that included several questions was used for gathering relevant information. Discussion groups with experts from each of the organisations were organised and conducted, and some of their documents were explored, in order to collect the necessary data to enable assessment of each site.

Table 5.14: The Organisations Teams – preliminary phase

<table>
<thead>
<tr>
<th>Interviewed teams within the three organisations</th>
<th>Site 1</th>
<th>Site 2</th>
<th>Site 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lieutenant General – RSADF Commander</td>
<td>Major General</td>
<td>Major General</td>
<td></td>
</tr>
<tr>
<td>Major General</td>
<td>Brigadier General*</td>
<td>Colonel*</td>
<td></td>
</tr>
<tr>
<td>Brigadier General</td>
<td>Colonel*</td>
<td>Colonel*</td>
<td></td>
</tr>
<tr>
<td>Colonel*</td>
<td>Captain</td>
<td>Major*</td>
<td></td>
</tr>
<tr>
<td>Lieutenant Colonel</td>
<td>2-Civilian Experts*</td>
<td>Captain*</td>
<td></td>
</tr>
<tr>
<td>Major*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-Civilian Experts*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total people interacted with during the visit is 19.

(*) Total people interviewed is 12, (8 Military + 4 Civilians), where all of them have no less than (7) years of experience in software development.

**Confidentiality:** Due to the type of work these organisations perform, confidentiality is a primary concern. Thus, their data, and the names of their sites have been withheld. Also, there is some data that was treated as confidential as per their request. Results of the case study will be published as an analysed amalgamation in the form of graphs, charts and/or tables with no identification. Participants were advised that all individual information would be kept private and unavailable to anyone other than specifically named researcher.

*However, the confidentiality imposed does not affect this study’s findings and recommendations.*

5.4.5 Result of the preliminary phase

The environments of the organisations were studied, in order to check their suitability regarding this case study. Consequently, a confidential report of the preliminary phase result was developed and sent to each of them, including all attachments needed (through a diplomatic mail from SACM), in order to allow them to comment on, and to approve a preliminary assessment and report.

The confidential report includes:

1. Comparison assessments and results.
2. Check-list, questions, answers, and comments.
3. Interviewed teams table.

The final comparison results (between the three organisations/sites), are presented in Table 5.15, which were based on the analysis of their feedback regarding the check-list questions. Where, each criterion worth (10 points), the totals are presented in the form of percentages within the last row.
Table 5.15: General criteria for selecting field study environment.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Site №1</th>
<th>Site №2</th>
<th>Site №3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reception of the Researcher by the organisation</td>
<td>Excellent</td>
<td>Best</td>
<td>Excellent</td>
</tr>
<tr>
<td>Organising experts team, to involve in the interview</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Degree of the team expertise regarding the research topic</td>
<td>Fair</td>
<td>Excellent</td>
<td>Best</td>
</tr>
<tr>
<td>Introduction of the organisation existing usable software development methods, which been provided by selected team</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Level of the team understandability to the researcher questions</td>
<td>Fair</td>
<td>Excellent</td>
<td>Best</td>
</tr>
<tr>
<td>Quality and precision of the answers given to the researcher questions during the visit</td>
<td>Fair</td>
<td>Excellent</td>
<td>Best</td>
</tr>
<tr>
<td>Quality and precision of the organisation feedback, that been included into the draft answer document</td>
<td>Good</td>
<td>Fair</td>
<td>Good</td>
</tr>
<tr>
<td>Access to required materials</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>General Support</td>
<td>Fair</td>
<td>Best</td>
<td>Good</td>
</tr>
<tr>
<td>Willingness of the case-study reception</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
<tr>
<td>Degree of the environment suitability to apply the case-study based on all information gathered during the visit</td>
<td>54 % (Not suitable)</td>
<td>78 % (Suitable)</td>
<td>80 % (Suitable)</td>
</tr>
</tbody>
</table>

Key: Bad (2)–Fair (4)–Good (6)–Excellent (8)–Best (10)

The overall assessment is:

• Site 1, did not have the right environment to conduct the case study.
• Site 2 and Site 3 qualified as suitable environments for the case study.

With a score of 80% for environmental suitability to the study context, Site (3) was selected for the case study.

A summary of the check-list findings for the preliminary phase are illustrated in Table 5.16, with Tables (E.57, E.58, and E.59) in Appendix E showing the complete check-list feedback and comments.

Table 5.16: Summary of the Preliminary Visit Findings.

<table>
<thead>
<tr>
<th>Preliminary Study investigation points</th>
<th>Site-1</th>
<th>Site-2</th>
<th>Site-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Organisations’ overall views about architecture.</td>
<td>Architecture is NOT their primary concern during software development life-cycle.</td>
<td>Architecture is NOT their primary concern during software development life-cycle. However, minimum architecture artefacts are developed by both (customer and contractor), within their new projects.</td>
<td>Architecture is NOT their primary concern during software development life-cycle. However, minimum architecture artefacts are developed by both (customer and contractor), within their new projects.</td>
</tr>
</tbody>
</table>
## 5.4. FIELD STUDY ANALYSIS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>None, most of the people working with the architecture are designers, and they are from contractor side.</td>
<td>Both systems are exist. Which is an advantage for this study.</td>
<td>Systems and sub-systems are tested after development is complete. They used scenarios, case studies, and scripts to evaluate systems. There is no evaluation on architectural level.</td>
<td>None. They just used primitive boxes and lines to communicate their concepts with each other.</td>
<td>Word, power point, and Visio.</td>
<td>None</td>
<td>No.</td>
<td>No.</td>
</tr>
<tr>
<td>3</td>
<td>None, most of the people working on architecture are designers, from both sides (customer and contractor).</td>
<td>Both systems are exist. Which is an advantage for this study.</td>
<td>Systems and sub-systems are tested after development is complete. They used scenarios, case studies, and scripts to evaluate systems. There is no evaluation on architectural level.</td>
<td>Primitive modelling used such as entity relationships, case study, and context diagrams.</td>
<td>Word, power point, and Visio.</td>
<td>None</td>
<td>No.</td>
<td>No.</td>
</tr>
<tr>
<td>4</td>
<td>None, most of the people working on architecture are designers, and they are from both sides (customer and contractor).</td>
<td>Both systems are exist. Which is an advantage for this study.</td>
<td>Systems and sub-systems are tested after development is complete. They used scenarios, case studies, and scripts to evaluate systems. There is no evaluation on architectural level.</td>
<td>More advanced SA description language used, such as UML.</td>
<td>Word, power point, Visio, Doors, and Rational Rose.</td>
<td>None</td>
<td>Partially following UML notation and DoDAF (1.5) framework.</td>
<td>No.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Software product documentation methods

| Site   | Software product documentation methods. | None. They used what they called “Packets” to document their software artefacts. Where their documentation is mostly mixed and not organised according to the development lifecycle; except the “Requirement”, which was clear and in a separate document. | None. Their software product documents were mostly mixed; the system architecture with the software architecture, and with design artefacts. No clear and organised methods been followed. Their “Requirement”, document was good, reviewed, and easy to follow. | Their documents, mostly organised, based on DoDAF (1.5) guidelines. However, most of their document does not pay enough attention for SA, but their documents still better than Sites (1 and 2). |

#### 5.4.6 Final Phase – Conducting the Study

The final phase was conducted over two months, between December 2013 – January 2014, and the findings are report in this section.

Based on the preliminary phase result, Site 3 was selected for the case study. However, due to logistics problems between Site 3 (the customer) and their vendor, many projects ceased after my preliminary phase visit. As a result, Site 3 was excluded from this study. Therefore, the case study was conducted with the second candidate (as illustrated in Table 5.15), which is Site 2.

In this phase, continuous interviews, meetings, and discussions were performed with 16 military people and 13 civilians, from both the hosting organisation and their contractors. The experiences of the 29 developers varied from 7 to 42 years, in the software development domain.

#### 5.4.7 Findings and Recommendations

During the field study, I (the researcher) interacted with government and industry organisations, through the hosting organisation, their local contractors, and their international contractors. According to the Capability Maturity Model Integration (CMMI) \(^3\) assessments published by Chrissis et al. [2007, pp 45], the capability levels of these organisations varied between level 1 (Initial), and level 4 (Qualitatively Managed) for both, their capability and maturity levels. Whereas, some of them do not have such assessments.

*Their project (system) development methods are:*

1. By external contractors, while the organisation (customer) works side by side with contractors, in order to insure that their requirements are satisfied.
2. In house projects, which are developed by the organisation experts.

I have investigated, discussed, and studied many projects and interviewed several developers, mostly from the organisation side. Projects involved in this study varied from small to large size, as described in

---

\(^3\)“The Capability Maturity Model Integration (CMMI) is a globally recognised set of best practices that enable organisations to improve performance, key capabilities, and critical business processes”, [2018]
5.4. FIELD STUDY ANALYSIS

Table 5.17. Some projects were already developed, and others were under development, which increased the value of the study, due to the ability to compare old with current projects, and how they varied, in regard to SA, SAE, and SA modelling methods.

Table 5.17: Investigated Projects during the case study.

<table>
<thead>
<tr>
<th>Project Size</th>
<th>Under development</th>
<th>Developed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Large</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2 Medium</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3 Small</td>
<td></td>
<td>47</td>
</tr>
</tbody>
</table>

Project size:
- $\$10,000 < Small < $\$200,000
- $\$200,001 < Medium < $\$2,000,000
- $\$2,000,001 < Large < $\$500,000,000

Over 100 documents from different projects were studied and compared. Advice and suggestions were given to the organisation regarding SA.

The document comparison criteria are:

1. Their availability?
2. Are they following standards?
3. Do they contain separate chapters for SA, or are they mixed with design chapters?
4. Are there any modelling methods used for describing their SA?
5. Are there any SAE methods that have been used and documented?
6. Overall quality of the documents?

Table 5.18 summarises the most important findings from the case study. The criteria listed in this table are based on the research objectives, and the findings of the two surveys reported in Chapters (4 and 5).
Table 5.18: Field study findings.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Findings</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Developers’ awareness regarding SA and SAE.</td>
<td>Most of the developers do not appreciate SA and SAE. Also, <em>(20 of the interviewed developers, out of 29)</em> do not know what SA and SAE means. This caused a problem during the study, in that a lot of my time was wasted trying to explain what SA and SAE meant and where it should be in the development life cycle. No training within the organisation focuses on SA and SAE.</td>
<td>1. Very clearly, developers are confused about the distinction between SA and design. 2. Lack of knowledge, maybe because most of the developers are programmers. 3. Lack of training</td>
</tr>
<tr>
<td>2 Developers’ awareness regarding SA documentation in particular.</td>
<td>Most of the developers do not know anything about SA documentation: how it should be done; and what artefacts should be produced. Some of the developers answers were <em>‘I just know that there is documentation for SA, and also there are no standards and guidelines on how SA should be described and documented’</em>!</td>
<td>1. Current developers’ lack of knowledge about SA documentation. 2. Significant shortage of human resources who understand system architecture in general, and SA in particular. 3. Lack of proper training that focuses on how to document SA.</td>
</tr>
<tr>
<td>3 Developers’ awareness regarding SA modelling and description languages.</td>
<td>Very few within the organisation know about UML, and they use it in their work individually. 90% of the interviewed developers <em>(26 out of 29)</em>, did not have knowledge about modelling languages, such as (SysML and ADLs). Actually for most of them it was the first time they heard of it. Developers were from different nationalities, and their qualifications are vary from Bachelor to PhD degrees.</td>
<td>1. Lack of training regarding SA description and modelling languages. 2. The organisation’s strategy does not give attention to modelling languages to describe SA within their projects. 3. Moreover, the organisation does not require modelling SA from the contractors’ side. 4. The lack of teaching of these technologies within educational institutes could be another factor that causes this problem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>SA description and SAE procedure within all projects.</strong></td>
<td></td>
</tr>
</tbody>
</table>
|   | There is clear neglect of SA and SAE. Most projects’ documentation was about detailed design. There are few about architecture. Also, their quality requirements are included within the statement of work. For example, the time for data transfer between two components within one of the projects was stated, as ‘the data shall be transferred within less than 60ms’. I have asked one of the developers, with PhD qualification in software domain, how they test such a requirement at SA level? His answer was ‘We never test it at that level, we test it after the system is finished, using test cases and procedures’.
<p>|   | Adding to all the above reasons, which persist here, the following reasons: 1. Most contractors try to decrease the amount of documentation of their projects if they can, because, such work is time consuming, and effort that could decreases their income. So, if the customer lacks the knowledge to require and trace appropriate levels of documentation, the contractors will take an advantage of the situation and produce a minimum documentation as possible. 2. Shortage of technical writers on both sides, organisation and the contractors. |
| 5 | <strong>Customer requirements regarding specific SA description or documentations.</strong> |
|   | The organisation (customer), does not require or specify any SA description methods or artefacts that need to be satisfied by the contractors. Thus, the organisation does not require any assessment for their potential system architectures. |
|   | 1. Lack of knowledge and experts in architectural level. 2. Sometimes the strategic plan, vision, and goals, of the organisation restricted by critical time to deliver some projects on specific time-frame, which could force them to minimise some of the documentation, as the case within some of their projects. |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td><strong>SA documentation for built in-house systems.</strong></td>
<td>Forty-seven (47-small size projects) developed within the organisations, by their own experts. I have investigated all the 47 projects regarding their documentation. I found only 8 documents that were created and available. All available documents, varied between requirements, design, interface prototyping, and testing procedures. So, the (47) working systems have only 8 documents that were created by individual developers, with low quality, and without following any documentation guidelines and standards.</td>
</tr>
<tr>
<td>7</td>
<td><strong>Available documentations regarding SA and SAE for developed systems.</strong></td>
<td>The best System/software architecture and design documents including SAE manuals and procedures were built in (May 07 1993), by Hughes Aircraft Company for C³ systems, which followed clear documentation procedures. All other projects neglect the importance of documenting SA and SAE. However, there is some SA documentation within other developed projects, but with low quality compared to the (old) documents created by Hughes company. Also, The new developed projects do not have any documents that evaluate or test SA in general.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Available documentation regarding SA and SAE for systems under development.</strong></td>
<td>Few documents were found for SA, but not for SAE. These documents are with low quality and a mix between SA description, design, and detailed design.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I have asked some developers about the reasons for such negligence. Their answers were as follows: “1. Shortage of human resources within their department. 2. Lack of proper training. 3. Shortage of SA experts and system/software design. 4. Most of the developers are programmers and they have no knowledge about SA. 5. The rapid and increased requests by other departments (within the organisation) for small systems to solve particular problems within their area, decreases their time to be able to document the product. 6. ‘The chain of stress’ from the highest management in the organisation down to the software department management, to deliver products ‘ASAP’, result in software’s and system’s without documentation.”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. For the most part, this absence and neglect of SA and SAE documentation, starts with the absence of organisations’ attention to these artefacts within their projects. However, the greater the organisation gives an attention to these artefacts, the greater for the contractors to satisfy their requirements. 2. There is an ethical side to this problem, which is lay on contractors. Contractors should be honest and tell the customer if he would such documentation, to help him to decide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same reasons listed in point (7) persist here. Also, some of the contractors (companies), do have shortage of expertise on the domain of SA and SAE.</td>
</tr>
<tr>
<td>9</td>
<td>The quality of SA documentation between old developed systems and systems under development provided by contractors.</td>
<td>Its obvious by comparing old and current projects documentations, that old project documentation was organised an followed clear documentation guidelines and procedures, with high quality of final outcome. The Figures and tables contained therein were sophisticated and had all notations needed to be understandable. Repetition of Figures and tables between SA documents and design documents was minimal, and differentiation between architectural diagrams and design diagrams was clear. The new projects’ documentation was incomplete, with low quality. Some SA documents are mixed with design documents, others do not have any SA documentation. The Figures and tables were repeated in different documents with no justification/ reason or with more details.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>10</td>
<td>Modelling software architecture within all projects.</td>
<td>There wasn’t any modelling techniques being used effectively to describe SA within any projects. There are individual attempts to model parts of SA for small size projects. These attempt did not evolve, to become part of any standard/ process.</td>
</tr>
</tbody>
</table>
5.5 An Experience Story

From 1998 until now (2018), I have participated in several large projects and many committees within the Saudi Ministry of Defence and Aviation (MODA)-(the customer), that were involved in developing new systems and improving existing ones, with costs of over 500 million US-dollars in total. All the contractors involved on these projects are internationals from different countries, and they are considered as large known companies in (military and security) systems. My work as a manager for communication, security, and tactical systems allowed me to closely inspect how SA was treated during the development process, from both sides (customer and contractors). A major advantage of this study is that I have been able to share my knowledge and recommendations with them.

This experience is strongly related to the research and it’s part of this study. Thus, in brief, I have summarised important points as follows:

1. In general, resistance merged with interest feelings were expressed by developers from both sides, when SA description, modelling, and evaluation methods were discussed within these projects and committees.
2. SA was treated with less priority and attention from both sides.
3. Primitive SA artefacts were developed (e.g. simple Component-and-Connector (CC)), ONLY to represent and communicate the ideas and concepts between both parties and between developers themselves (e.g. managers, system engineers, designers, and programmers).
4. Most developers are programmers, designers, system engineers, and project managers. No professional architects are available on-site. Architects are called in only when needed.
5. SPs are utilised without considering their QAs.
6. SA Modelling techniques and languages (e.g. AADL, UML-2.0, SysML, Object-Role Modelling (ORM), and Service-Oriented Modelling Framework (SOMF)), are not used, except in some primitive use cases, sequence diagrams, entity relationship diagrams, and context diagrams with the UML-1.5 language and Visio tool.
7. SA evaluation methods (qualitative and quantitative) are not used within architectural artefacts. However, Scenario based methods are used to test the systems after their completion. They normally perform five tests: component test, system test, integration test, in-plant test and on-site test.
8. SA frameworks are not utilised or followed.
9. SA documentation is not separated, but are merged within system engineering documents.
10. As per discussion with two project managers, regarding SA; I asked them to develop project software architectures using ISO-42010 standard, and AADL, UML-2.0, and SysML languages. Also, to apply DoDAF-2.0 views, and to evaluate architectures both qualitatively and quantitatively. Furthermore, they were asked to document their activities via a proper standardised mechanism. Budget was an obstacle, as over 500,000 US-dollars was required in order to perform the above requirements, which did not fall within the financed statement-of-work requirements. Also, the customer’s limited knowledge, time-frames, and carelessness regarding SA did not help to address these requirements. The customer’s responses to these requirements, were along these lines, “if
5.6. CONCLUSION

the system works properly without these requirements, we are happy”. So, what happens if the customer wants to improve these systems in the future (and they will)? They will need to hire another company to do so, which is much more costly than developing systems with proper methods, including all necessary documents and source code.

11. Unfortunately, both the customer and contractor developers overall knowledge and concern regarding SA aspects are too limited.

The above eleven points support the findings of this chapter. The gap between government and industry on one side, and academia on the other, concerning SA, is wide! The utilisation of SA as a 'concept' within software development is applied by all. But, whereas academics consider SA as a part of the working systems by utilising new languages and methods, some government and industry sectors (as is the case here) consider SA to be a concept on paper, just to help people involved to communicate and understand each other. This gap could be improved by doing more empirical research, and by employing new SA languages and methods in government and industry sectors.

5.6 Conclusion

SA has been investigated from different views: its description, modelling, documentation, evaluation, and utilisation. Two important methods were used (questionnaire and field study), which were complementary in regard to some findings. Both methods’ analyses showed very interesting findings. One important finding is the difference between the effort that has been put into SA current research and literature, and the ignorance utilisation of SA in the real world, as well as, the importance of QAs within software products, which is undeniable.

Chapter 4 identified the fact that most people do NOT consider QAs when they choose their styles as a part of any architecture artefact. One reason could be the absence of proper tools that would help them to evaluate their SA, including contained styles.

The findings of this chapter indicate that documentation and tools have a major role to play in the utilisation of SPs and models during SA description. Furthermore, the availability of tools and effective documentation for SPs, and SA languages might result in the creation of an effective framework to improve quantitative evaluation methods, and to increase the utilisation of patterns and modelling techniques, which could improve the final product’s qualities.

Furthermore, this study discussed some factors that could hinder the utilisation of models during architecture description, as well as, SAE quantitative methods, in order to discover any issues or gaps that could be incorporated into future research plan, and to contribute to SA domain.

Also, the results of this study identified some important factors that could impact (positively/or negatively) SA from different angles as discusses earlier, such as:

- The awareness of new technologies that support the SA domain.
- Formality level of SA description.
- Utilisation of modelling languages and SPs, when describing SA.
- Following reliable standards and frameworks for creating a proper SA.
- Tools support.
- Realisation of the difference between SA and design among software engineers.
• SA evaluation methods, and their automation possibilities.
• Employment of SPs and SA (processes, methods, languages, ...etc) in real world projects.

The gap between academia and industry regarding the overall thinking and utilisation concerning SA, increases the need for more research and education, in order to introduce clear and reliable methods to describe, utilise, and document SA. As a result, the SA domain could be improved, and become easy to understand, easy to be traceable to the code level, easy to be evaluated, easy to be automated, and easy to be documented in a proper fashion.

There are aspects of SA, including awareness, documentation, analyses, modelling etc, which deter SA from being widely used within industry and government. Architects having adequate SA knowledge seem few and far between, and virtually no practicing software engineer seems to have been adequately trained, to be motivated to study or apply SA for the purposes of improving software project outcomes.

Consequently, I recommend more research with different methods (e.g., empirical research), in order to, unearth the importance of design rationale documentation, resolve each one of the obstacle factors, and discuss any aspects that have NOT been covered sufficiently by this study.
Chapter 6

The RCS as a Case Study and Promoting the *Moreno et al.* [2008] Approach

Maintaining architectural integrity is not easy. Architectural quality requires investment and discipline.

— Dr. Bill Curtis, senior vice president and chief scientist at CAST

*Mistrik et al.* [2014, pp xxvi]
6.1 Introduction

The main aim of this chapter is to validate and complement the previous chapters’ findings through two main sections, where each section has its own preamble. First, is the RCS examination (Section 6.2), which illustrates the strong relationships between SA, SPs, QAs, and their effectiveness on each other. Introduced in the Second section is an abstract theoretical concept for SA evaluation (Section 6.3), based on improving the Moreno et al. [2008] concept, in order to (contribute to, and enhance) software development processes.

6.2 RCS under Examination-Intro

This section is a discussion and report on the architectural analysis results for the Intelligent Reference Architecture-Real-time Control System (RCS) as a case study that was built upon the knowledge of the issues surrounding SA, SPs, and QAs relationships within discussed in previous chapters. RCS was created at National Institute of Standards and Technology (NIST), Albus et al. [1996]. The modified version of the RCS, called 4D-RCS, was developed by Albus [1997] in order to provide a theoretical foundation for engineering, and architecture, for unmanned ground vehicles.¹ The architecture of (RCS/4D-RCS) is considered a reference architecture in the domain and has been proven to be applicable in several applications, see Albus et al. [2001] and Meystel et al. [2002].

The focus of this analysis is to uncover any employed SPs within the (RCS/4D-RCS) architecture and, consequently, identify any related QAs possessed by those patterns. Nonetheless, the idea beyond this work is to prove the concept regarding the strong relationships between SA, SPs, and QAs, as reported in Chapters (3, 4, and 5). Also, to demonstrate their effectiveness on each other, by exposing the conflicts between QAs within the RCS architecture.

The discussion and analyses of the RCS architecture reported in this section were performed based upon the following references:

1. Albus [1997], Albus et al. [2001], Meystel et al. [2002], Albus [2002], and Russell [2011], which describe the RCS architecture.
2. SEI team, Bass et al. [1998], Bass et al. [2003], and Bass et al. [2013] books, which describe architectural styles.
3. The references listed in Section 3.2 that describe many patterns/styles and their QAs.
5. The relationships database between SPs and QAs reported in Chapter 3.

According to Yin [2013], “A text is used to describe and analyse the case. You may augment the text with tables as well as with charts graphics, pictures, and maps, depending upon the depth of the case study”, which normally is used for a single-case study. This report on the RCS case study is written based on the guidelines published by Runeson et al. [2008] and Yin [2013].

¹This section concerns architecture artefacts that are owned by both RCS-versions (I, II, III) and 4D-RCS. Thus, they are considered to be of no difference during this examination. The rest of this report will use ‘RCS’ as the main name of the targeted architecture. Any variations within RCS versions will be indicated when required.
6.2. RCS UNDER EXAMINATION-INTRO

6.2.1 Case Study Process

Based on the detailed descriptions of the RCS that have been provided by the references mentioned in the previous section, the study has been carried out using four processes as illustrated in Figure 6.1.

Figure 6.1: RCS architecture analysis steps.

The following analyses will neither explain nor describe the styles individual concepts. However, the relationship between both styles and architecture will be discussed in detail.

6.2.2 Discussion and Findings

This section is composed of two sub-sections that provide a critique of the styles incorporated within the RCS architecture. Both sub-sections refer to the same figures to support the arguments presented; Figures (6.2, 6.3, and 6.4) are the main ones for this analysis. These figures illustrate the RCS overall architecture and node structure, including the Observe, Orient, Decide and Act (OODA) loop, invented by Boyd1977.

Each node contains four main computing modules, which are:

1. Behaviour Generation (BG), for planning and control.
2. World Model (WM), which includes the Knowledge Database (KD), for store knowledge, and predict. Where the knowledge in (KD) includes processes, entities, attributes, maps, tasks, events, and states.
3. Sensory Processing (SPr), for sensing, filter, recognise, detect, and interpret.
4. Value Judgement (VJ), for compute cost, benefit, and uncertainty attributes.

An example of the RCS overall architecture (for unmanned ground vehicle) is provided in Figure 6.2. Data in the KD is shared between WM modules within the nodes, and at the same level within the same sub-tree (see thickened black lines). On the right are examples of the functional features of the BG activates at each level (e.g. 5 second plans). On the left, are examples of the data kinds and maps mentioned by the WM in the KD at each level (e.g. surfaces). The interface on the right, provides input to, and output from, the node elements. The OODA loop starts from the sensors, processed through the
nodes, then data/actions forwarded to the actuators; subsequently, sensors observe the actuators actions then process the data again etc, Albus [2002].

Furthermore, Figure 6.3 shows the abstract relationships and activities that take place between the RCS node elements, for both evaluation and execution phases. Whereas, Figure 6.4 demonstrates the node elements relationships with more detail, including the planning loop and the feedback loop.

Figure 6.2: RCS overall architecture (for Military Application), after, Finkelstein [2008].

Figure 6.3: Functional relationships between modules of Elementary Loop of Functioning (ELF), after Meystel et al. [2002].
6.2. RCS UNDER EXAMINATION-INTRO

6.2.2.1 RCS architecture and BCK styles

Discussion, analysis, and interpretations of the RCS architecture are demonstrated in Table 6.1, based on SEI styles\(^2\) descriptions, in order to reveal the existing styles and QAs within the RCS architecture.

Table 6.1: BCK styles and QAs within RCS architecture.

<table>
<thead>
<tr>
<th>BCK Styles</th>
<th>Discussion and Analysis</th>
</tr>
</thead>
</table>
| Call and Return → Layered       | • Layered style helps to structure a large system that requires decompositions, as is the case in RCS. The whole RCS architecture is an organised hierarchy, where computational nodes are arranged in multi-layered styles like the posts in a military organisation.  
• The relation between Job Assignor (JA) and his Agents is organised based on the layered style. Also the typical planning process has a layered architecture style.  
• The transition between levels of abstraction results within SP\textsubscript{r} imaging processes and grouping entities form a layered style; (e.g. pixel entities grouped into list entities and list entities grouped into surface entities etc). Each grouping process collects entities and events up to the higher level entities and events, as illustrated in Figure F.2.  
• Outputs from the bottom layer of BG modules drive actuator and the inputs to the bottom SP\textsubscript{r} layer modules conveys sensor data. “The communication system conveys commands from BG modules to their subordinates and returns status”.  
• The inner structure of the Executer (EX) within the BG shows an obvious layered style starting by receiving input from plan selector until becoming input task to lower JA in the next BG module or a command to the Actuator at the lowest level, as described in Albus et al. [2001, pp261-262]. |
| Independent component → Event System (Implicit and Explicit) Invocation | • Event-based styles have been applied almost right across the whole RCS architecture, and its clear specifically in RCS typical node at level (i), where event could be top-down such as in the Planner (PL) component, where the JA received a task command event from the higher level then starts to process the spatial task decomposition into small tasks and pass it to agents, which start to do their assigned job, then trigger another component within the system to do its job based on the received information until the event passes the desired action command to the actuator, as explained in the sensory processing chapters by Albus et al. [2001] and Meyste et al. [2002], see Figure 6.6 (see page 223).  
• Other sequences could be bottom-up events, which start (for example) from perceived sensory input to the SP\textsubscript{r} components, in order to process their signal. Consequently, triggering another components and passing information to them (such as perceived object, event to the VJ, or the higher SP\textsubscript{r} signal). The “Top-down” defines significant goals and priorities by the BG. “Bottom-up” defines unexpected, unusual, or threatening actions by SP\textsubscript{r}. Both sequences are employed within RCS architecture.  
• Also the event system could be a horizontal sequence between the component in the same level or the same layer such as the relations between the SP\textsubscript{r} and WM, where the SP\textsubscript{r} triggers the WM, requesting some information/data/image from Knowledge Database (KD), in order to process the image recognition and comparison. |

\(^2\)BCK (Bass, Clements, and Kazman) styles in Table 6.1 are the same as SEI (Software Engineering Institute) styles. BCK is short for the ‘software architecture in practice series’ authors.
The inner structure of the EX, within the BG shows an obvious event style with layered structure. Each EX sub-module for each agent closes a feedback control for its agent. Also, for a discrete event system, each EX sub-module increments its controller from one step to another, which is detected by SPPr and reported again to the EX module by WM.

- RCS is a network or event system, where entities and components trigger each other, requesting/ordering/sending commands or data. While, observed entities, events and situations are sent by SP to VJ. More explanation of the event style appears in the mechanism of MVC pattern, which is just an instance of an event driven architecture, noting that MVC is not a design pattern. MVC is considered for higher level structures and it supports modifiability.

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Pipes and Filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data centered → (Blackboard and Repository)</td>
<td></td>
</tr>
<tr>
<td>&quot;The Blackboard is an architectural pattern useful for problems for non-deterministic solution strategies. Several specialised components “assemble” their knowledge to build a possibly partial solution, coordinated by a central controller” (Ortega et al. [2008]), as is the case between the RCS ‘node elements’. Each component is specialised for solving a particular part of the overall task, such as is the case in the RCS architecture. KD within WM forms an active repository (not passive), which means the Blackboard style is in all RCS nodes. The KD is always updated and maintained by WM in each node.</td>
<td></td>
</tr>
<tr>
<td>Part of KD functions (as with Blackboard style) is to send predicted information to SPPr, while the WM and VJ update the KD information, Albus et al. [2001, pp. 198].</td>
<td></td>
</tr>
<tr>
<td>Also, KD represents a passive repository that sends information to JA, EX and Scheduler (SC).</td>
<td></td>
</tr>
<tr>
<td>KD has a Short Term Memory (STM) that forms an interface/ or buffer between immediate experience and Long Term Memory (LTM). The STM persists until it is overwritten. It acts like dynamic Random Access Memory (RAM). Whereas, LTM furnishes a repository of information that can accumulate and be retained over a lifetime. LTM differs from STM primarily in the fact that the information endures over time, Meystel et al. [2002].</td>
<td></td>
</tr>
<tr>
<td>Pipes and filters style provides a structure for systems which process a stream of data such as is the case in RCS. In general, RCS component relationships are based on processing input and producing output to other components. For example, the lowest SPPr reads the sensor signal, then processes it as a ‘filter’ and then sends it to the higher SPPr, or other components through ‘pipes’ and so on. At each level, the SPPr modules filter and process information derived from subordinates, Figures F.1 and 6.5.</td>
<td></td>
</tr>
<tr>
<td>BG receives commands from a higher level, processes it as a ‘filter’ using (JA, SC, and PL), then sends the information as input to EX (Figure 6.6). The Executer components use the EXs outputs as an input to the lower level of BG for further processing etc. Note that the inner structure of EX and the PL constructed as pipes and filters style each component processes its input and sends its output to other components, etc., Meystel et al. [2002, pp. 303, 319, and 333].</td>
<td></td>
</tr>
<tr>
<td>To conclude, the RCS uses the pipes and filters style between its components to process the data, and to transfer the information between its components, because the pipes and filter style emphasises the incremental transformation of data by successive components, which is the case between several layers, nodes, and components within its architecture.</td>
<td></td>
</tr>
</tbody>
</table>
6.2. RCS UNDER EXAMINATION-INTRO

• RCS could be considered as a set of communicating processes, in terms of communications between the nodes internal components such as WM and BG. Also, the communications between nodes throughout the RCS system follow the communicating processes style. These communications could represent the concepts of clients and servers relationship (style), with the client as requester component and the server as provider component, of the information. Nodes communicate with each other by means of messaging, implemented using the Neutral Messaging Language (NML), which supports a variety of communication protocols. This communication system provides an inter-communication system between modules within the node. Queries and task ‘Status’ are communicated from BG modules to WM. Retrieval of information is back from WM to BG. While, predicted sensory data is sent from WM to SP, where updated data is sent from SP to WM, Figure 6.3.

Figure 6.4: RCS computational node Inner-structure, after Finkelstein [2008].

Note that (conceptually) there are some other styles/patterns within the RCS architecture, which are not included in the BCK styles collection. Two of these ‘styles’ worthy of mention are: Presentation-Abstraction-Control (PAC) and the Microkernel styles. PAC is an interactive (software) system in the form of a hierarchy of cooperating agents. Every agent is responsible for a specific task. An agent denotes an information processing component, including receivers and transmitters. The style is obvious at each node within the RCS, between higher nodes and their subordinates, including the components (BG, WM, SP and VJ) and their subordinates. Also, the PAC style is evident between JA and its agents. The RCS explicitly allows for exchange of information between organisational units and agents at the same level, or even at different levels. Command and status reports flow only between supervisor and subordinates, but queries, replies, requests, and broadcasting of information are used to convey information between any of the units or agents within the entire RCS, Meystel et al. [2002].

Furthermore, the structure of the RCS supports modifiability by applying the Microkernel concept
Figure 6.5: A top-level Elementary Loop of Functioning, with two control levels (nodes), after Meys et al. [2002].

(style), which separates a minimal function core from extended functionality. Also, because of its architectural qualities, the RCS is able to be easily adapted to different environment and apparent system types.

Nodes in the RCS are functionally independent. This allows the system to be more flexible and modifiable. Also, adding and removing nodes is easy, and any changes do not impact the whole system.

In addition, the RCS architecture is heterogeneous in that the system is not limited to running on specific hardware or software platform. These styles are described in Bass et al. [1998, pp 102.]

The hierarchically and simultaneously heterogeneous styles are obvious within RCS, whereas locationally they are not. However, the overall RCS architecture works as an organisational, computational, and behavioural hierarchy control system, Albus et al. [2001, pp. 158]. Both hierarchical and horizontal relationship styles within the RCS architecture are illustrated in Figures F.1 and 6.5. The structure is hierarchical (multilevel hierarchical architecture). Commands and status feedback flows are hierarchically up and down throughout a BG chain of command, and SP - WM flows of information. This is not classical hierarchy architecture, it is more of a tree architecture, which implements the PAC style concept that is used as a hierarchical structure of components or agents.

It’s a multi-scale relational structure, which means vertical branches have horizontal connections that make it a more complex system. The architecture is horizontal, because the data is shared horizontally between heterogeneous modules at the same level. At each level the architecture is horizontally connected by communications pathways between modules (BG, WM, SP, VJ and KD) within the same node, and between nodes at the same level, especially with the same command tree. Each agent acts as a participant
of a group in scheduling and organising with peers at the same level, while simultaneously acting superior to its subordinate unit at the next lower level. The RCS architecture is formed by five main styles (see dominating styles below) that are interrelated with each other, communicate with each other, and overlap each other, which makes it a mix of all these styles. Therefore, based on the analysis above, the RCS architecture could be considered as a hierarchically and simultaneously heterogeneous style.

Based on the above discussion and analysis, the RCS architecture consists of several different styles, but some styles dominate more than others within the architecture. The dominating styles within the RCS architecture are:

1. Layered.
2. Pipes and Filters.
3. Event System.
4. Blackboard.
5. Hierarchically and Simultaneously Heterogeneous Styles.

### 6.2.2.2 The RCS architecture and Mellor Styles

In this Section, observations and analysis concerning the RCS architecture are discussed based on the style descriptions reported by Mellor [2009], as illustrated in Table 6.2. There are some similarities between Mellor [2009] styles and BCK styles, which are mentioned during the analysis without repetition.
Table 6.2: *Mellor* [2009] styles and QAs within RCS architecture.

<table>
<thead>
<tr>
<th>S. Mellor Styles</th>
<th>Discussion and Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event based</td>
<td>• The analysis of the Event System (Implicit and Explicit) invocation style in Table 6.1 is pertinent here, because it is the same style, and will not be repeated.</td>
</tr>
<tr>
<td>Monitor and control</td>
<td>Based on Stephen Mellor’s recorded session-3 (ANU course, 2010), this style should be periodical. Its data should be in one place for fast access and it could be monitored or controlled (not both) in some cases. This style has related control links and loops, in order to set control points or measures, read values, make comparisons or displays, and perform computations. So, based on this description, the following points relate to the RCS architecture:</td>
</tr>
<tr>
<td></td>
<td>• The BIG PICTURE of the RCS architecture exhibits monitor and control style, where the system reads the sensor values and processes information within all layers and components, then controls the actuators with a signal, then other sensory information comes, and so on, in a monitor and control loop. All these processes are periodically timed. Furthermore, the operator interface function is to provide monitors at all levels of the system, including lowest level, by sending a command such as (Halt system), then receives feedback from the system in the operator interface display, which closes monitor and control loops between the system hardware components feedback (sensors and actuators) and the operator interface through the system components and layers.</td>
</tr>
<tr>
<td></td>
<td>• Monitor and control style reads values from hardware for comparison or display as Mellor stated. The structure of the nodes as parent - child relations are considered the same in concept, where each node sends a control command to the lower level node to be processed, and receive information from lower level to be displayed or compared.</td>
</tr>
<tr>
<td></td>
<td>• The SPr receives predicted image data from WM to make comparison with the observed image, then sends the information to WM to update the KD with the current state of the world, and forward the world state to the EX. EX receives this feedback through the comparison and monitoring module, in order to compute the difference between the estimated state feedback from WM and the current planned sub-goal. Also, in order to compare and monitor sub-module progress to achieve the current sub-goal and feed, the action will be forwarded to the lower level or (the actuator) in the lowest level. This process closes loops as per the monitor and control style.</td>
</tr>
</tbody>
</table>
### 6.2. RCS UNDER EXAMINATION-INTRO

- Each EX sub-module closes a feedback control for its agent and increments its controller from one step to another, while detected by SPr, and reports again to the EX module by WM. This makes it a control style.
- A close control loop within planner entities is clear, where the sequential searches of the spatial and temporal plans are evaluated and compared in order to choose the best plan; if not, the search continues.
- The relationship between SPr and sensors could be considered as an independent monitor style. Whereas, the relationship between EX and actuator forms an independent control style. *This concept could be useful, sometimes, as a separation of concerns, or decoupling, which usually helps to increase maintainability, flexibility, and modifiability.*

<table>
<thead>
<tr>
<th>Transporters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transporters style involves the transfer of data from one point to another with no change of the data contents. The main concept, based on Mellor’s description (session-3), is that the output of a component should be the same input, and should reflect the correct status of the real world. Transporters contain computational processes, decomposition of the data, and reassembling of the data.</td>
</tr>
</tbody>
</table>
- The overall RCS architecture sends data from layer to layer, from node to node, from module to module, from component to component which means the RCS mechanism follows a transporters style.
- WM with its KD acts as a transporter style. RCS modules (SPr, WM, and VJ) update KD with all information needed for the system, while KD provides information to different RCS components as they receive it. So, it is transferring data from components to another (from one point to another). For example, WM updates its KD short memory with current state by SPr, and then sends this information to EX, *Albus et al.* [2001].
- JA receives task commands from its parent and decomposes it into subtasks to its agents with no changes, so the data is then processed and transferred from component to component within each agent, until producing an output task. These data movements within the planner are considered as transporter style.
- SPr processes sensor information from the real world object, then forwards it to higher levels, and so on. This illustrates a transporter style to represent the same real world object on the operator display, *Meystel et al.* [2002].
- The transporters approach is close to the pipes and filters architecture style with one important constraint, no content should be changed within the transporter. Whereas filters can enrich, refine or transform its input data. Furthermore, the transporters style provides a structure for systems that process a stream of data such as is the case in 4D/RCS. In general the RCS component relationships are based on processing input and producing output to other components. For example, the lowest SPr reads the sensor signal and processes it as a transporter, then sends it to the higher SPr or other component, etc.
- *As a result, the RCS architecture uses transporter style between its components to process the data and transfer the information between its components.*
This style maintains a picture of the real world, requests and updates that picture, does some computations, and sends responses to the outside world. All kinds of simulators form a transaction style. (Mellor session-3)

The RCS architecture is a transactions style, as illustrated by its components being in an hierarchy, such as SP, WM, VJ, BG, and KD, which are distributed throughout the set of RCS Nodes. At all levels of RCS architecture, SP and WM processes work together to maintain the world picture in the KD for the same node, keeping it current and consistent. The KD in each node contains the information required to support the BG and SP processes and requests within that node.

Also, KD information contains the resolution over the range in space and time required by decision making, which is controlled by VJ and BG processes in that node. BG completes the cycle with a response command output (based on the computational process done by all the node components) to be sent to the outside world (actuator) to do their tasks. All requests, responses, updates, simulations, and computational processes within the RCS components as a whole, and its nodes in particular, satisfy Mellor transaction style characteristics.

Consider the closed loop from BG-Planner to WM-simulator to VJ evaluator, with the evaluation feedback which results when VJ sends it to BG-Planner. To illustrate, a request from BG to WM, in order to simulate the plan and forward the result of the simulation (predicted plan) to the VJ for evaluation, with the response returned to the planner. This process forms a transaction style.

Maintaining a picture of the world by updating the KD information mechanism is considered a transaction style. For example, SP compares the observed image with the predicted image and WM uses the differences to update the KD, which makes it a transaction style.

Requesting information by SP (such as Predicted Image) and by BG (such as current state) and the knowledge from KD for the planning process, through the WM satisfies the transaction style features.
6.2.3 Manifestation of QAs within RCS architecture

The overall realisations for some of the QAs that are embedded within the RCS architecture are discussed in this section, noting that the RCS architecture encompasses more QAs than what are presented here.

The QAs that have been mentioned in the RCS documents and other respective references are listed below:

1. Functionality
2. Modifiability
3. Reliability
4. Extensibility
5. Portability
6. Efficiency
7. Integrability
8. Interdependency
9. Performance
10. Space and time
11. High precision
12. Availability
13. Interchangeability
14. Maintainability
15. Flexibility
16. Customisability
17. Exchangeability
18. Fault tolerance
19. Scalability

The rest of this section explains the most visible QAs within the RCS system. “Architecture is critical to the realisation of many of the qualities of interest in a system, and the qualities should be designed in and evaluated at the architecture level”, Bass et al. [1998].

1. Functionality:
The RCS system defines functional modules at each level, each one embodies a set of responsibilities and priorities, and the structure of the nodes and the components coordination of the whole RCS architecture support the functionality and the ability of the system to complete its mission. With all its components and knowledge sources (VJ, WM and KD), the RCS system has been developed to support a vehicle that is capable of carrying out missions with low data rate communications, and to operate for extended periods without human intervention. The detection processes for errors at SPri and the computational process for errors at EX help the system to do its functions in the most efficient ways. This is explained more in Fault tolerance (at point 6 below).

2. Modifiability:
RCS is a reference architecture model that has to be adaptable to new technology in the future, so it must be a modifiable architecture. Modifications to the system, extensibility, and adapting to new environments are all forms of modifiability. Modifiability could be the QA that most closely aligns with the architecture. RCS contains an independent component architecture, which consists of a number of computational nodes and agents that communicate with each other through messages. In general, the whole RCS architecture supports modifiability, starting from the nodes construction to the smallest module within a node. The RCS architecture is based on an independent components style where the basic component is the node, and each node comprises different modules, BG, SPri, VJ, WM, and KD. Also, each one of these modules contains other internal modules. Making modifications to the RCS is done by adding or deleting modules, extending the system to add new nodes to do new functions, and to assign new responsibility to those nodes without extensive effort to modify the significant parts of the system being required.
Also, all elements within the RCS node may have an operator interface within the interface layer, which makes it easy to add new operational interfaces for new functions, and new elements within the node modules without affecting the whole structure. Using Layers and Pipes-and-Filters styles
gives the architecture the major benefit of recombination, enabling greater flexibility and modifi-
ability.

It is worth mentioning that one of the BCK business’ qualities, which is related to ‘Time to market’,
is the ability to insert components or products, such as into COTS, depends on the decomposition of
the system into well-defined components, such as the case with RCS. On this basis, an architecture
such as RCS is useful now and could be ascendant in the future.

3. **Portability**:

It’s the ability of the RCS to run under different computing environments (Hardware and Software),
which makes it a valuable reference model. Local dependencies between layers and decoupling
components, usually ‘constrain’ the effect of code changes, hardware changes, special data formats,
operating systems, etc., which often affects only the layer to which changes are made. This supports
portability and testability as well, since you can test particular layers or components independently
of other components in the system. The RCS architecture gives the opportunity to implement the
same architecture in different ways, because it’s a reference and general architecture.

These modifiability and portability quality attributes are evidenced by the fact that the RCS has
been used in several quite different applications, such as horizontal machine workstations, a con-
trol systems for the US postal service, an autonomous mobile vehicle, and others, *Albus* [2002, pp
26]. These applications have different hardware and software environments; however, the success
of RCS in these applications is clear evidence of the high portability of RCS to run in different
environments. Also, the encapsulation principle within the nodes and modules in the RCS layers,
affords an adaptable and replaceable architecture across different operating environments. That
gives an abstract interface to the environment. The lower layer allows different sensors and actu-
ators to be implemented, and it does not specify the types of sensor or actuator that RCS should
use.

4. **Re-Usability**:

Reference models, such as RCS, should be reusable, and allow and help the architect/designer
to solve similar conceptual problems in different contexts and in different ways. According to
its flexibility and intelligent construction, the RCS architecture is considered as a proof of the
successes of the re-usability.

According to *Buschmann et al.* [1996] (POSA-V1), if an individual layer embodies a well-defined
abstraction and documented interface, the layer can be used in different contexts. The RCS sys-
tem has a well defined layered and components. All its nodes have been clearly identified and
decomposed. Also, using filters to support recombination, results in re-use of filter components
and support interchangeability as quality attributes of the RCS architecture.

5. **Performance**:

Performance concerns timing and event concerns, such as interrupters, messages, requests, etc.,
The RCS is a real-time control system, while the 4D/RCS is a reference model that integrates RCS
with the 4-D approach to dynamic machine vision, in order to get the best performance for the
Unmanned Ground Vehicle (UGV) in the battlefield, where time affects life and death or winner
and loser. Each layer within the RCS should respond to the received events in a specific time, based
on the functionality of the layer and the responsibilities of its components.

The RCS architecture has been structured with high performance aspects. Also, it is organised,
such that the JA decomposes each input command task into a set of jobs, which will be assigned to
SC’s for subordinates BG units. This is a spatial decomposition to achieve better performance.
The planning of behaviour, the control of action, and the focus of computational resources, help functional modules at each level to process limited specific manageable tasks, which reduces the complexity factor at each level, and, as a consequence, improving the responses among components and eliminating the overload in order to have a high-performance system.

As soon as a level one planner submits a plan to EX, it begins replanning immediately. At a minimum re-planning must be completed before the current plan is completely executed. "The RCS BG process is designed to support both real-time and off-line planning. Versions (I and II) of the RCS used off-line planning exclusively", Albus et al. [2001, pp 181]. Whereas RCS V-III supports both processing times.

6. **Reliability, Fault Tolerance and Availability:**

These three QAs are combined because they are strongly related to each other. A system needs the ability to maintain its functionality reliably (reliability) and to ensure correct behaviour in the event of errors (fault tolerance), in order to be a functional system. Failure should be prevented, detected whenever it occurs, and should be recovered as fast as possible (Availability).

According to Bass et al. [2013], availability tactics are fault detection, fault recovery and fault prevention. **Redundancy** is to be considered as a tactic to achieve specific QAs. Redundancy of the components, at every level within the RCS architecture, helps to achieve reliability, fault tolerance, and availability, which are required by most military and manufacturing systems.

The RCS system is reliable, because one of the VJ functions is to assesses the reliability, and generate rewards and punishments. Moreover, within its nodes at each level, feedback loops are closed to provide reactive behaviour. Also, at each level in the RCS hierarchy, SPs detect errors and makes comparisons between what is expected, and what is observed. Small errors are used to update the WM. Large errors indicate the need to assume new entities or objects for more discrepancies between predicted and observed sensory input. EX within BG computes errors at each level between the current planned sub-goal (i.e. desired state) and the observed state of the world. Also, EX computes error compensates and modifies output command parameters to correct errors.

Furthermore, RCS errors are detected at lower-levels first, where they can be addressed most quickly. The lower level sensory processing functions are the first to detect states and events, that indicate problems or emergency conditions, such as velocity, acceleration forces, etc.

It is worth noticing that the use of independent styles (event style, for instance) within the RCS architecture helps the reverse engineering process (even if the original source code is unavailable), which can be done by extracting architecture events from a running system via instrumentation, such as monitoring middle-ware. *The event based architecture and the call tree can be extracted by debugging software, Donohoe [1999].* QAs are important aspects within architectural styles and design patterns.

In some cases, it is hard to evaluate an entire architecture’s quality characteristics, especially if it’s a heterogeneous architecture, such as the RCS, but it is not impossible. This evaluation difficulty is because of the existence of discrepancies between QAs that are possessed by different styles that form the overall architecture. However, the recommended evaluation mechanism is to divide the overall architecture into smaller sub-architectures that could be evaluated easily.

For example, the conflict relationships between the identified architecture styles embedded within the RCS and their QAs are summarised in Table 6.3. However, there is no qualitative or quantitative evidence provided by those references. While, most of the current SP/styles and QA relationships are based upon the authors observations and experience.
Chapter 6. The RCS as a Case Study and Promoting the …

Table 6.3 demonstrates some QA conflicts by summarising five styles that are included within the RCS architecture (as an example). The highlighted (sky-blue) columns include QAs which are affected, supported or hindered, by styles utilised by the RCS architecture. For example, the Layers style supports five QAs and hinders two. Consequently, the complexity of SAE and the relationship between SA, SPs, and QAs are apparent, which motivates researchers to develop an approach that accommodates individual pattern evaluation with an overall architecture evaluation.

Note: The patterns sequence numbers within the database (explained in Chapter 3) are shown between parenthesis ‘()’, under each pattern name.

In addition, creating a system architecture (software and hardware) knowing in advance what styles/patterns could be useful, and what QAs they could support or hinder, is much better than creating an architecture missing this knowledge, which could lead to an architecture that utilises patterns/styles with conflicts in different aspects. As a result, the final product could be unexpected and with unsatisfied requirements. However, predicting the overall qualities of an architecture as a whole, is an arduous task but beneficial.

Table 6.3: QAs trade-offs within RCS Styles.

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Changeability</th>
<th>Efficiency</th>
<th>Maintainability</th>
<th>Portability</th>
<th>Integrability</th>
<th>Usability</th>
<th>Reusability</th>
<th>Availability</th>
<th>Efficiency (dealing with large amount of data)</th>
<th>Exchangeability</th>
<th>Simplicity</th>
<th>Extensibility</th>
<th>Flexibility</th>
<th>Interdependency</th>
<th>Modifiability</th>
<th>Performance</th>
<th>Scalability (size, configuration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackboard (71)</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Layers (31)</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Microkernel (171)</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>PAC (145)</td>
<td>↑</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Pipes &amp; Filters (53)</td>
<td>↑</td>
<td>↑</td>
<td>↓</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
</tbody>
</table>

Key:

↑ = Support
↓ = Hinder

To conclude this main section, an appraisal process involving four phases is followed (as illustrated in Figure 6.1), using critical analysis during the entire process.

The first three phases include the following activities:

(a) Study, absorb, and fully understand the concepts and detailed description for the RCS architecture, and SEI and Mellor styles, from different resources.

(b) Comparison of the information derived, in order to discover the styles that are encompassed
6.3. Incipient Concept to Promote Moreno et al. [2008] approach

Intro

Latency is the delay encountered in communicating a message from an input point to a required output point. Processing Time is the amount of time taken to process a given request by a system, without including the message transfer time (latency time). Consequently, \((\text{Latency} + \text{Processing Time} = \text{Response Time} \Rightarrow \text{Performance of a System})\), Iqbal [2015].

The approach presented by Moreno et al. [2008] (explained in Section 2.5.3.6) was developed to measure performance. A new preliminary concept to extend and improve the Moreno et al approach has been identified, in order to minimise its limitations, gain from its advantages, and by the use of SP and QA tactics support the SAE processes.

Figure 6.7: The link between the entire RCS appraisal process and Chapter 3 findings.

6.3.1 Patterns and QAs conceptual schema

One of software development’s most important issues is to deliver products with required ‘qualities’. The intent of the approach presented here is to provide an evaluation concept for SA, including...
its patterns/styles, to ensure that quality issues are addressed immediately during the first stages of the software development life cycle.

Evaluating architecture should go hand-in-hand with developing architecture during the software life cycle. The representation of the general relationship between the model driven engineering path and the model driven analysis path, is illustrated in Figure 6.8, Becker [2008].

The main goal of using patterns is to design architecture with known QAs, Buschmann et al. [1996]. Various techniques that capture and represent QA and their relations to architectural solutions have been assessed and discussed in Chapter 2, and they form an important source of knowledge regarding the development of this new concept.

![Diagram](image)

Figure 6.8: Modelling Development Engineering (MDE) and Model Driven Analysis. Becker [2008].

The concept described in this section is similar to the patterns re-factoring concept. Kerievsky [2005] used patterns as a starting point, but when the ‘the pattern’ did not work as is, he and some others started using re-factored patterns, referred to here as the patterns re-factoring concept.

For example, the Model View Controller (MVC) pattern has been widely used, but cannot be executed properly even with a clear concept of MVC in mind, mainly because of its implementation complexity, as established by Kerievsky.

Thus, the effort and findings reported in the previous chapters regarding SPs inspired the idea of, and provided the motivation for, developing this new conceptual paradigm.

### 6.3.1.1 Conceptual schema description and steps

In order to make the conceptual schema easy to understand, the textual description is represented graphically in Figure 6.9.

*The following steps explain the conceptual framework:

1. Understand and document the ‘context of use’ for an architectural pattern and its variety of implementations by the evaluators. For example, the Check Point pattern information reported in Table 3.2, represents a minimum sample of the required data, Reussner et al. [2005, p154].

2. Identify the supported and hindered QAs that will be affected by the selected pattern. The database
6.3. INCIPIENT CONCEPT TO PROMOTE cite[Moreno, G.Moreno et al. …

![Conceptual patterns evaluation schema](image)

Figure 6.9: Conceptual patterns evaluation schema.

developed during this research and reported in Chapter 3 is the starting source.

3. Identify the main forces associated with the selected patterns. The forces are “a way of exploring the indications and contra-indications for the pattern”, Fowler [2006]. The presence of the forces within the pattern’s context are illustrated in Figure 6.10. Also, Table 3.2, shows some examples of the forces that drive the developers to use the Check-Point pattern.

![Forces within Pattern ingredients](image)

Figure 6.10: The forces within Pattern ingredients, after Tešanovic [2005].

4. Use and enhance the applicable existing set of architectural tactics, or develop new tactics if needed for the intended problem domain (e.g. Table 3.2). QA analysis studies, such as the identified tactics by Bass et al. [2013] or properties by Babar et al. [2005] (discussed in Chapter 2), may help to develop the best-fit evolution methods, theories, scenarios, and metrics to be used by Analysis Model during the evolution process. For example, if the security of an architecture is to be evaluated, different tactics may be used to define measurable metrics. The security tactics’ path from Bass et al. [2013] demonstrates the idea:

security ⇒ recovering from attack ⇒ identification ⇒ audit trail.
Consequently, Audit Trail evaluation, from ISO 25010, can be used for external or internal security metrics. Figure 6.11 represents ISO security metrics as an example. Also, it shows that the right metrics can be identified starting from the root of the security tactics (e.g. Recovering from Attack).

![Security Tactics - Bass, 2003](image)

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Purpose of the metrics</th>
<th>Method of application</th>
<th>Measurement, formula and data element computations</th>
<th>Interpretation of measured value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access auditability</td>
<td>How to complete the audit trail concerning the user access to the system and data?</td>
<td>Evaluate the amount of accesses that the system recorded in the access.</td>
<td>(X = \frac{B}{A}); where (A): potential number of accesses to system and data; (B): Actual number done during evaluation.</td>
<td>(0 \leq X \leq 1). The closer to 1, the better.</td>
</tr>
<tr>
<td>Access auditability</td>
<td>How auditable is access login?</td>
<td>Count the number of access types that are being logged correctly as in the specifications and compare with the number of access types that are required to be logged in the specifications.</td>
<td>(A - Number) of access types that are being logged as in the specifications; (B - Number) of access types required to be logged in the specifications.</td>
<td>(0 \leq X \leq 1). The closer to 1, the more auditable.</td>
</tr>
</tbody>
</table>

ISO Metrics For Auditability

<table>
<thead>
<tr>
<th>External Metric</th>
<th>Internal Metric</th>
</tr>
</thead>
</table>

Figure 6.11: Derived evaluation metrics starting from the right tactics.

5. Develop the necessary *scenarios* for functionality checking, or for evaluation purposes, concerning (SA/or SP). *Scenarios* are brief narratives of expected or anticipated system uses from both user and developer views and they provide a look at how the system satisfies quality attributes in various use contexts*, Kazman et al. [1996].

As an example, a few scenarios for the Check-Point pattern are presented in Table 3.2. Figure 6.12, illustrates the main activities of the scenario-based approach during SA analysis. Scenario-based methods are a broad topic; however, the main scenario methods were discussed earlier in Chapter 2. Also, there are many references that explain the development process for scenario generation and
6.3. INCIPIENT CONCEPT TO PROMOTE cite[Moreno, G. Moreno et al. …

how they should be applied, such as Armour et al. [2001], Clements et al. [2002a], Lewis [2016], and SEI [2018], which could be useful to construct the scenarios during this step.

Figure 6.12: Brief of the scenario-based dependencies and activities during SAE, after Kazman et al. [1996].

6. ACME and ACME-studio are used (as an example of tools and languages utilisation) to check the structural conformance of the architecture components, in order to produce the input for the QAs reasoning framework. ACME as an architectural language has been discussed earlier in Section 2.3.4. Figure F.5, represents a sample of Pipe and Filter family architecture using the ACME-studio tool. Other languages and tools can be used as required, such as AADL, SysML, and Artisan tool.

Figure 6.13: Pipe and Filter family illustrated through ACME-Studio tool.

7. The QA reasoning framework (Figure 6.9) includes: Input architecture, Reasoning components
CHAPTER 6. THE RCS AS A CASE STUDY AND PROMOTING THE …

(yellow box), Output (analysis results), where Reasoning components has its relations and properties and a collection of QA models that can belong to each QA reasoning mechanism.

The following three phases explain the concept framework’s main processes:

(a) Create the input architecture – Develop an architecture (or pattern concept), using for example ACME language and its tool, SysML and Artisan tool, or other languages and tools. Defining domain-specific architectural styles and plug-in analysis tools that may be invoked by the analyst is possible here, because ACME-studio is written as a plug-in to the Eclipse framework. However, the structural constraints that are already built into some tools such as ACME-studio and Artisan must be satisfied by a pattern structure. It should also satisfy the analytical constraints, which depend upon a specific quality reasoning framework that needs to be added into the pattern architectural description using the same tool.

The pattern descriptions (e.g., Figure F.5) that have been created using ACME-studio are the input for the Interpretation component, which transform the architecture/pattern description into a specific quality model corresponding to the QA that has been selected for evaluation. The limitations and challenges of the Interpretation elements are reported briefly during the analysis of Moreno et al. [2008] performance model in Section 2.5.3.6.

(b) Analytical phase – The interpretation component extracts the Quality Model from the original architecture/pattern description to be used by the evolution model (Analysis Model). The Analysis Model uses the framework repository, which contains analytical theories, metrics and scenarios. Expert knowledge is needed to create a proper interpretation component to be able to produce sufficient information within the quality model for each required QA.

The Moreno et al. [2008] approach used an additional step, which was to transfer the original architecture into an ICM meta-model that forms the input to the reasoning framework (interpretation component). This is to simplify their original architecture for analysis processes, which could cause a problem as explained later in the limitation section.

(c) Analysis results – Based on the repository information identified for each quality, such as the evaluation theory, scenarios, simulations, and metrics, the information on the quality model is used by the Analysis model for evaluation processes. Scenario triggers are also included in this model. This analysis model could use different scenarios that had been included in the repository and can activate them by using scenario triggers (see next section).

6.3.1.2 Prospect of the conceptual schema

A classification of a set of wide-ranging scenarios for which a reasoning framework is relevant is called a scenario trigger. These scenarios are recommended to be included within the patterns description table.

Formalising different analytical theories, metrics, and scenarios, and subsequently categorising them based on a specific quality and storing them in a repository, will result in reusable data that will be produced by professionals from different domains. As a result, productivity should be increased, the cost and time consumption should be minimised, and the automation for the evaluation process will be encouraged.

In addition, the ATAM approach could be used as a hybrid technique for both scenarios and pre-existing questions. Also, using the ABAS library (created by SEI group) could be very helpful in creating an analysis model and in choosing the right evaluation methods.

236
An external simulator could be hooked to ACME-Studio/or Artisan tool for evaluation purposes. In order to ensure that the simulator receives the appropriate information, a repository can serve as a source of knowledge, where different types of information, scenarios, metrics, or simulation packages/scripts are included. This repository can be modified according to any required assessment. If in case, a new scenario or simulation is produced, it should be tested for applicability, and then added to the repository. The information explained earlier within Chapter 3 database, Sub-section 6.3.1.1, and Table 6.4 are examples of the different types of data that could be useful and included within the repository.
Table 6.4: Reliability/availability evaluation, methods, data, and the generic framework, after Grunske 2007.

<table>
<thead>
<tr>
<th>Method</th>
<th>Encapsulated Evaluation Method</th>
<th>Operational Profile</th>
<th>Composition Algorithm</th>
<th>Evaluation Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamlet, Mason, Woit</td>
<td>Reliability measures, independent from the operational profile of components, profile mappings are used to obtain the reliability measures in a specific deployment context.</td>
<td>Operational profiles at the system level (also known as trail profiles).</td>
<td>Composition of the evaluation models based on the system-control flow.</td>
<td>The system reliability is calculated based on traditional reliability equations (extended by equations for conditional cases and loops).</td>
</tr>
<tr>
<td>Yacoub, Ammar</td>
<td>Dynamic reliability metrics.</td>
<td>Description of the operation profile with sequence diagrams.</td>
<td>Generation of component dependency graphs, that describe probabilistic call sequences as Markov models.</td>
<td>Assessment of the reliability-based risk of a component by traversing the Component Dependency Graph (CDG).</td>
</tr>
<tr>
<td>Reussner, Poernomo, Schmidt</td>
<td>Parametric contracts, a generalisation of the design-by-contract principle based on the Quality of Service Modelling Language (QML) Froland et al. 1998</td>
<td>Service effect automata, that describe the call probabilities of services, these service effect automata are also used to describe the environment of a single component.</td>
<td>Composition of service effect automata + identification of the accepted language (traces) of the composed service effect automaton.</td>
<td>For each trace the reliability of a service can be determined with traditional methods, the final reliability is the sum of the individual trace reliabilities.</td>
</tr>
</tbody>
</table>
To achieve a required QA, it’s recommended to create the architecture by utilising patterns from the same domain (e.g. Pattern Language). As an example, a catalogue of related patterns that are known as a Security Pattern language presents a solution description to some common security problems. However, different patterns’ weaknesses and strengths should be known, well-documented, and summarised so as to be addressed and accessible by software designers. For instance, Figure 6.14 shows a subset of the security pattern language for system access control and their relationships,

![Figure 6.14: Subset of the security pattern language for system access control and their relationships.](image)

By using the patterns in Figure 6.14, the outlines of a design process can be seen.

**To evaluate a pattern against QAs using this approach, the suggested process is to:**

1. Select the target pattern that needs to be evaluated.
2. Identify the related QAs by using reliable references, such as the relational database (explained in Chapter 3), or from a set of architectural tactics.
3. Create the pattern architecture using a proper language, such as ACME, SysML or AADL;
4. Choose a reasoning framework concerned with the quality under evaluation;
5. Input the pattern architecture description into the interpretation component to produce an appropriate specific quality model for the evaluation process;
6. Choose suitable evaluation methods and/or scenarios (general and concrete with mapping mechanism) from repository for analysis process;
7. Finally, evaluate the quality model by using the analysis model, which incorporates the necessary and appropriate evaluation methods, and then transform the results into readable data.

In order to demonstrate the processes of the conceptual model, Figure 6.15 shows a simple example of measuring object coupling, through the utilisation of SysML profile and Artisan tool. The output of the three main phases that are described in Section 6.3.1.1 are depicted within the figure. Also, the structural and analytical constraints included, were based on the Tibermacine [2014] study for measuring object coupling. The result (line graph) illustrated within the example is based upon Coupling Between Objects (CBO) metric, as defined by Chidamber et al. [1994]. Taking into account that the interpretation and quality components are not developed, their functionality concepts are clearly described within the diagram.
Figure 6.15: Example of the model processes for (coupling measure), by utilising SysML and Artisan-tool.
Development and implementation of this new concept is not as easy as it might seem. It needs a lot of work and research. This general approach provides a conceptual basis for evaluating QAs within architecture/patterns, by demonstrating the concept outline for the main components, and the model processes within Figures 6.9 and 6.15, which could be able to aid during the software development and evaluation processes. Full implementation of the outlined concept is recommended for future research.

The extension of this abstract concept to include multiple QAs reasoning frameworks, solid repository, fully developed models, and test procedures processes are dependent upon further research to uncover the best and latest modelling languages, theories, integration mechanisms, tools, evaluation approaches that could support the evolution of this concept in the future.

6.3.2 Conceptual schema limitations

The QAs and SPs issues should be taken into account during architecture design, because it is also difficult and costly to retrofit a required quality into an implemented system. A gap is noticed between analysis engineering and architecture engineering knowledge. The use of different sources to identify and capture architecturally sensitive quality knowledge and present it in an easy and understandable format might help to close the gap between both disciplines analysis and architecture engineering.

However, this first attempt for the proposed solution has its limitations. So, the use of advanced architecture description languages is important for developing better evaluation models. For example, ACME as a language description for this proposed approach will result in ACME disadvantages being imposed on the approach as a whole. Where Section 2.3.4, discussed ACME in general; the following drawbacks of ACME emphasise the importance of language selection:

- ACME dynamic configuration of components and connectors is not strongly supported.
- ACME describes a weak typing system with a small permanent set of types, such as the seven architectural elements described earlier in section 2.3.4 and its core ontology and data types do not support associations that need new data types, taking into account that data types are an important aspect for evaluating software architecture from different aspects, Kamal et al. [2007].
- The absence of better tools that support ACME and architecture description standardisation, make it hard to develop and implement all aspects of the proposed solution components. Other languages, profiles, frameworks, and tool could be more advanced and more promising, such as UML(2.5), SysML(1.4), AADL-2, DoDAF(2.02), UPDM(2.1), Artisan-tool, as illustrated in Figure 6.15.

Notwithstanding the drawbacks of ACME, the new concept does have disadvantages:

1. The use of interpretation components in the approach makes it difficult to generalise the solution to evaluate different QAs, because each QA needs specific interpretation components, which may make it costly and time consuming.
2. The components of the conceptual model are not fully developed and tested.

To conclude, this is the first trial to establish a concept to assess the quality characteristics of an architecture through the evaluation of SPs. Included is a description of the complexity of the quality
assessment and the necessity to overcome this complexity during the architecture evaluation. Finally, the proposed solution is still theoretical and is not fully developed or tested for applicability to the problem domain. As a result, further research needs to be done.

6.4 Conclusion

The aim of the two case studies described in this Chapter 6 was to validate and complement the findings from the review of literature and the examinations of current practice, as established via two surveys and a field study, in relation to SAs, SPs and QAs and their relationships. The first case study, based on an appraisal of part of the RCS Architecture, demonstrated i) that there are existing conflicts within SP and SA relationships, and ii) how to identify existing QAs within an architecture and any relationship conflicts between them.

The second case study sought to establish if the Moreno et al. [2008] performance model could be better utilised in the context of SA evaluation. An improved Moreno model was created through incorporation of six additional components and its use demonstrated via application across a number of different aspects (patterns, tactics, matrices, languages, and tool utilisation).

In considering the outcomes of these two case studies, I realised that there was the potential for two new concepts to be developed in relation to i) the evaluation of system architecture, and ii) quantitative evaluation of software architecture. My initial exploration of these two new concepts, referred to as SysAE and SAQEF, is provided in Appendix G as the basis for future research.
Part III

Wrapping up
Discussion and conclusion

Understanding of mind and brain will enable the creation of new species of intelligent machine systems that can generate economic wealth on a scale hitherto unimaginable.

Albus et al. [2001, pp 377]
7.1 Introduction

The main objective of the research reported in this thesis was to explore SA description, modelling, and evaluation techniques across several engineering interdisciplin ary domains, in order to enhance and improve the SA arena. This objective has been accomplished through the contributions reported in the previous chapters.

The rest of this chapter provides a summary of the contributions, findings, limitations, and recommendations arising from the research, before wrapping up with suggestions for potential future research.

7.2 Summary of the research contribution

Different aspects of SA (including patterns) have been investigated and discussed, through the study of, and comparisons between, various current SP resources, SA description languages, and SAE approaches, with a view to answering two important questions:

1. What effect do description languages, standardisation, evaluation methods, modelling techniques and their documentation have on ‘SPs and SA’ utilisation and evaluation?
2. How can ‘SPs and SA’ utilisation and evaluation be improved, including minimising the effect of any hindrances?

The grey box below summarises the answer to these questions.

By undertaking various ‘inspections’ of the state-of-the-art material, significant limitations, gaps, and conflicting information were uncovered. Also identified was the existence of some strong relationships between several areas (within the software engineering field), such as SPs, QAs, SA, evaluation methods, description languages, modelling techniques, standardisation, and documentation procedures, which have an effect on ‘SPs and SA’ utilisation and evaluation.

First perceptions of the literature indicated inconsistencies. Thus, a critique style investigation and analysis of some of the current reliable references was performed to verify the inconsistencies. As a result, several problems such as differences, inconsistencies, omissions, gaps, and limitations, were proven to exist within SP descriptions. A partial solution to the identified problems is illustrated through Chapter 3, together with recommendations to examine the other identified problems.

Two surveys and a field study were executed to confirm (or not) these findings and to answer some aspects of the research questions. The approaches used did confirm the findings and views regarding targeted areas (Chapters 4 and 5). This confirmation provided some insight into user views concerning both ‘SPs and SA’, through the current state-of-the-art materials, and the usefulness of those materials in their own work, including views on their refusal/willingness to embrace SPs, SA, or modelling methods more often. Several problems have been identified through both Chapters 4 and 5, and solutions to these problems recommended (e.g. more teaching for both SPs and SA in academia).

Proof of the existing relationships between SA, SAE, SPs, and QAs and their impact
on each other were accomplished through a case study in Chapter 6. The conflicts between qualities within the RCS architecture support the findings from previous chapters, and also provided proof of the usability advantage of the ‘SPs-QAs’ database that was developed (Chapter 3). Moreover, the development of the conceptual evaluation model (including its example in Chapter 6) illustrates how we can improve the way of thinking concerning SA evaluation.

In Chapter 3, I proposed a partial solution to the problems that have been identified regarding SPs, such as their description, conflicts, minimal usage, and documentation methods, (e.g. instead of searching for specific pattern relationships with quality attributes within thousands of pages, researcher can find this relationships within a seconds by typing the pattern name in the database). I argued that improving the documentation of SPs and their relationships with QAs could also improve the evaluation of SP; hopefully to improve the SA domain in general.

The main problems discovered concerning SPs were the inconsistencies and absence of common standard documentation procedures, description methodologies, categorisation variance, inconsistent definitions, and discrepancies in their relationship with QAs. As a consequence there is confusion among users, leading to difficulties with pattern selection, unknown quality (before use) of employed patterns and discouragement of developers to utilise them. The solution proposed was to create a common database that contains all patterns from six different reliable resources, including: their definitions, categorisations, and relationship with QAs, and with an easy search method in order to help pattern users quickly select required patterns, compare their definitions, view their category, and to see which qualities they support or hinder. For example, the method and documentation of SPs can be see in Checkpoint pattern description Table 3.2, and by gathering several definitions in one place for comparisons; while standardisation are shown by the use of ISO-42010 standard as the base for the (SPs-QAs ) relationship. Whereas, relationship Table 6.3 in Chapter 6 between identified SPs and QAs illustrates, how we extracted these relationships from the database during the RCS analysis. However, it is important to recognise that the database information needs a continuous work, which includes improvement and update in regular basis, in order to be useful and current.

Further research within the same area was recommended, by means of applying a survey, which was carried out and reported in Chapter 4.

The survey undertaken gathered software engineers’ opinions regarding SP’s current documentation, description procedures, and their utilisation. A primitive database was developed to facilitate the questionnaire’s preliminary analysis, followed by a full analysis of the information obtained. The analysis results support Chapter 3 findings (mentioned above), and identified further problems concerning the SP domain:

• Conflict between pattern descriptions within the current literature.
• Lack of integration mechanisms between patterns.
• The lack of pattern exposure to the required audience through academic institutions.
• Absence of a scientific method to prove pattern relationships with QAs.
• The difficulties of pattern selection processes, without knowing which QAs they do support before employing them.
• Documenting patterns with no common standard or methodology.
• Lack of effort and research to improve all the aspects mentioned in the above points.
The survey questions include some proposed solutions (in the form of statements–within the last four questions), which were strongly supported by many of the questionnaire respondents. One of these proposed solutions is to create a database for SPs; a database was developed as part of this study, and reported on in Chapter 3.

Further, the analysis findings indicated the need for more research concerning SA and SAE, which is addressed in Chapter 5. Two different methods were employed, in order to seek further information regarding SA descriptions, documentation, and evaluation methods. The first method is ‘a questionnaire’, which examined the most important aspects that are related to SA and SAE. The second method is ‘a field study’ that was professionally organised and carried out on-site within a large government organisation in Saudi Arabia. The information gathered from both methods (the survey and the field study) was analysed scientifically. Most of the findings from both methods support each other, such as:

- Few developers used SA modelling languages.
- The developers wide nescience regarding SA description languages, and evaluation methods.

The finding indicate that SA description, modelling, utilisation, and evaluation are important, but not yet practised enough by developers on real life projects, as was expected.

Also, and unfortunately, software engineers awareness and usage of SA and SAE methods are low. Contributing factors identified include:

- Deficiency of ‘software engineers’ knowledge concerning modelling techniques, and SA in general;
- Limited teaching of these approaches in academia;
- Few SA and SAE documents of a high quality standard, and with clear full examples of both approaches, exist and are openly available to developers; and
- Lack of tool support.

Chapter 5 also contains the report on the current architectural procedures and documentation from different large-scale projects derived from a field study. Problems, proposed solutions, recommendations, and limitations from this study are provided. Part of the field study involved interacting with developers from three different organisations, working side by side with them on diverse projects, which varied from (small to large) in size. As a result, my practical knowledge, and skills surrounding SA are greatly enhanced.

Based on my research findings in Chapters 2-to-5, I proposed two important “pillars of practice” for implementation into the research problem domain (SA arena), as in Chapter 6:

1. Examination and analysis of an existing and reliable SA reference model, called the (Real-time Control System (RCS)), Albus et al. [2001], in order to “show the influential dependencies”, between SPs, SA, SAE, and QAs, which were realised during the previous chapters’ arguments and findings. The trade-offs, and conflicts between QAs within the RCS architecture were demonstrated.

2. Developing a preliminary concept, to improve an existing performance evaluation model, the Moreno et al. [2008] performance model, in order to generalise that model to be applicable to several QAs, not just for evaluating performance.

Six components were added to the the Moreno et al. [2008] model, in order to form a potentially
7.3 LIMITATIONS AND DRAWBACKS OF THE CONTRIBUTION

An improved model. How the “improved” model works is explained in Sections (6.3.1.1 and 6.3.1.2), and examples of patterns (e.g. Pipe and Filter), tactics (e.g. Security), matrices (e.g. audit trail), and tool utilisation (e.g. ACME-Studio) illustrated.

Furthermore, another example was illustrated through a description of objects coupling evaluation processes, for a small portion of the RCS architecture (RCS Node), in order to demonstrate and simplify the model process and concept. The resulting “improved” model, considered as an initial concept given its limitations, is explained in Section 6.3.1.2. It was while undertaking this particular evaluation that I began to think about developing a better framework and models to evaluate SA. These thoughts are explained briefly in Section 7.4, and Appendix G.

7.3 Limitations and Drawbacks of the contribution

In 1998, Sir. Morwenna Griffiths denied the existence of perfection for any research in any field of study, stating “there is no hope of doing a perfect research”, Griffiths [1998]. I concur with his statement, because research is expected to produce a novel concept or improve existing ones, resolve new problems or existing ones, in order to promote the life of mankind, which could be affected by many (controllable and uncontrollable) factors, such as, type of research, data type, reliability of the data, knowledge of the researchers, tools support, time, cost, research environment, etc.

Therefore, this research has its limitations, as is the case with practically any other research. Some of the research limitations were discussed within each chapter, thus, this section will briefly explain the general limitations and drawbacks, as follows:

- Both questionnaires, reported on in Chapters 4 and 5, inherited the common disadvantages of surveys, such as differences in interpretation, dishonest answers, non-conscientious responses, hard to convey feelings, and questions that were hard to analyse. Moreover, both questionnaires tried to cover more topics than perhaps they should have, such as patterns documentation, utilisation, and their relationship with QAs (Chapter 4) and SA description languages, modelling techniques, and SAE (Chapter 5). As a result, both questionnaires while comprehensive and advantageous to their domains, were perhaps not focused enough concerning each of their sub-topics.

- Regarding the field study in Chapter 5, the involved organisations are not familiar with such research, or researchers coming from outside their organisations, which added some challenges throughout the study processes from organising-to-performing. I was the first person to undertake such a study in these organisations. However, the final outcome, and the results and recommendation gained are important and beneficial for both the organisations and research. The study generally lacked quantitative results and precise measures, due to the scattered and pressed environments of many projects, time limitation, and the confidentiality of the projects and sites; hence, limiting the declaration of some information.

- The analysis of the RCS and the evaluation model initial concept that are reported in Chapter 6, are described with all necessary processes, examples, and arguments. However, the attempt to improve the Moreno et al. [2008] model, shows some limitations that are mostly in accordance with the improved version, as described in Section 6.3.2. Consequently, another SA generalised evaluation framework idea has evolved that could overcome some of the Moreno et al. [2008] model disadvantages, which are explained briefly in Section 7.4.
7.4 Recommendations and Future work

The challenges and gaps that are influencing different aspects of SA have been demonstrated in this thesis, together with suggested solutions for some of these challenges, such as:

- Gathering different SA quantitative metrics in one repository for public use.
- Developing automated quantitative methodology to evaluate SA.
- Developing a comprehensive reliable tool for SAE purposes.

There are a few recommendations that could be given, in order to avoid some of the limitations identified. Also, there are opportunities to extend this study and to promote some of its concepts, models, suggestions, and solutions. So, this section illustrates some of these directions.

1. I recommend narrowing the scope in any similar study, or to divide the overall scope into sub-scopes, so as to be able to cover all the study objectives from all angles, and to be fully able to develop necessary artefacts – especially if time is constrained.
2. Narrowing the scope of questions is also recommended – ‘especially’ for questionnaires similar to the ones presented in Chapters 4 and 5.
3. It is advisable to refine and improve the SPs database, with a more concrete and better architecture, interface, and with more functionality. This should include: completing the information of (SPs and QAs) tables, such as the forces, general scenarios, and metrics. The database design should be able to interface with external models and/or tools.
4. Regarding the model illustrated in Chapter 6, full development, implementation, and testing of the model components, its interface mechanisms with the database illustrated in Chapter 3, and requesting and processing procedures for the scenarios, matrices, etc, for each quality characteristic (from the database), is recommended.

The trial of improving the Moreno et al. [2008] model (Chapter 6), leads to another direction of research as follows:

5. To develop, implement, and test, a general standard SA evaluation approach and models that require two important components; a System Architecture Evaluation (SysAE) concept, and Software Architecture Quantitative Evaluation Framework (SAQEF)). Both, concepts are explained briefly in Appendix G, with some examples.

The following components should be developed and implemented, in order to fulfil both theoretical and concept goals:

- Full SPs – QAs database, with integration mechanism to SysAE.
- Full SAQEF framework.
- Architecture data extraction mechanism.
- Full SysAE Components.
- Result visualisation mechanism for SAQEF output.
- Internal integration mechanism between SysAE components.
- External integration mechanism between SysAE and the external environment.
7.5. OVERALL CONCLUSION

The SysAE and SAQEF approaches need to be up-to-date with current technology. Thus, they should be designed with self-upgrading capabilities.

The knowledge encompassed in this thesis could serve as a foundation to facilitate the development of the above seven components, which also could be accomplished by utilising current professional and reliable languages, standard, framework, and tools to develop all artefacts (e.g. AADL, SysML language, DoDAF profile, ISO-42010 quality standard, and the Artisan-studio tool).

In order for the proposed evaluation system to fully achieve its objectives and to be applicable in industry, it should be able to answer questions, such as:

1. Are there any constraints for SA (description, notations, and semantics)/ or how it’s presented, to be evaluated by the (SysAE and SAQEF) approaches?
2. How efficiently, and to what levels, will the SysAE components evaluate complex systems?
3. Is the SysAE approach inbuilt with enough capabilities to evaluate a range of system architectures from different and/or multi-domains?
4. How best can the SysAE approach builds its components with capabilities that meet future industry needs?

7.5 Overall conclusion

A combination of investigatory techniques, proposed solutions, and conceptual models, have been developed and presented within this study, which leads to novel findings that contribute to the SA and SAE fields.

In addition, the uniqueness of this study, as compared to other research found in the current literature and industry, relies upon its process, diversity of methods used, and the knowledge that has been merged/linked between different interdisciplinary domains, which should improve SA understanding among the software engineering community.

There is slight knowledge and employment of SPs, SA, and SAE methods by current software engineers.

Thus, the key outcomes of this study are:

1. The development, description, and documentation of SPs and SA, need to be improved to be used by the software community.
2. SPs, SA description languages, modelling method, evaluation techniques, and QAs, should be considered during SA development.
3. The academic sector needs to include: SPs, SA, SAE within their curriculums, in order to educate future students about the value of knowing these subjects, and to prepare them to be able to (contribute and enhance) the overall software domain.
4. Industry and government sectors need to hold workshops for their employees to achieve the same objective as in point (3).
5. More improvement of SA description tool and SAE tool could increase the deployment of SA concepts within software projects in the future.
6. More research needs to be done, on how to read/interpret SA for its (specific and general)
evaluation purposes.

7. A full example of SA development processes, including: all associated artefacts, verification (method and results), and validation (method and results), needs to be developed and published. Also, the example should be explained and documented completely based on a standard, or a clear procedure, and to be open source, in order to increase software engineering community’s knowledge regarding the SA domain.

The clear roadmap for future research, presented by Section 7.4, is a ‘useful’ outcome of this study, which needs to be attended to with utmost care.

7.6 Closing remarks

Keeping this work within a relevant context, when carrying-out more research in the future, will enhance the viability and visibility of the SPs concept, QAs evaluation, and overall SA domain. The overall arguments, contributions, findings, and suggested future thoughts, could be considered as an interdisciplinary approach that will help in steering researchers to improve and resolve some of the problems identified here. The long term aim being to help stakeholders select the best architecture candidate for their intended systems, through their realisation of the full architectural potential and limitations, which will lead to the production of better systems.
Bibliography

(2018).


Iqbal, Z. (June 2015). Latency, processing time, and response time.


The end
Appendices
Appendix

Publications

This Appendix introduces the thesis published papers in its original format. The order of the papers presented here, based their chapters order. The relation between the papers in this appendix and the chapters in the main thesis body, as follows:

1. First paper related to Chapter 3
2. Second paper related to Chapter 4
3. Third and fourth papers related to Chapter 5
SOFTWARE PATTERNS vs QUALITY ATTRIBUTES
(INVESTIGATION APPROACH)

Hassan Almari¹, Clive Boughton²

Australian National University (ANU), Research School of Computer Science, Canberra, Australia¹,²

Abstract: The development of software patterns (SPs) is aimed at providing a reliable and reusable framework for resolving similar problems within distinct contexts. To accomplish this objective competently, it is imperative to document these patterns effectively to facilitate the comprehension of their concepts to users, thereby encouraging their use over and over again. Thus, the documentation of patterns needs to explicitly explain their relationship with the quality attributes (QAs) that they support, or hinder, in order to satisfy the implementation of stakeholders’ requirements. The variation in patterns descriptions in contemporary literature renders the explanation of the above relationship complex and difficult to follow. This eventually deters developers from employing patterns or causes them to overlook their QAs. Either of these scenarios may result in significant expense in terms of development time and cost, and/or attaining required system quality. This paper tries to address the aforementioned problem by comparing and analysing six well-known software pattern resources, pinpointing the aspects of variation amongst authors descriptions, which lead to different relationships between patterns and QAs, which in fact cause confusion among users. Once the variance concept amongst these six resources in terms of terminology and description has been addressed, we derive a relationship matrix between the software patterns (included in these resources) and the standard ISO-9126 QAs. We believe that this research work is a positive contribution to the enhancement of techniques for documenting software patterns. It further helps improve pattern selection by users via improved prediction of output quality. Thus, to provide a reliable method for maintaining and representing the research work, we have created a database application that identifies the above relationship. This database also includes discrepancies among the documentation approaches of the six resources that we have studied, as well as the variance in pattern categorisations and terminologies. The pattern database should also serve future research endeavours. This research study received a positive response as per the findings of a questionnaire aimed at software professionals and based on the context of the preceding problem. 97 percent of the participants, from six different nations, answering the questionnaire supported this study.

Keywords: Software engineering, Software architectures, Patterns, Quality concepts, Quality analysis and evaluation, Documentation

I. INTRODUCTION

Currently, most software pattern resources describe patterns based on the authors’ experiences and observations. Some of these resources have pointed explicitly to the relationship between each pattern and their (apparent) quality attributes QAs, i.e., [1], [2] while others do not; i.e., [3] and [4]. However, there are a few works analysing the identification of the relationships between software patterns and quality attributes in a scientific methodology [5], based on measurements and metrics, such as that of the work done by [6] and [7]. The former mentioned work of Kim/Garlan used (Alloy-Analysers) as a tool. They tried to create models using some patterns, and to evaluate some quality characteristics. Their work focused on mapping rules between architectures and models, while maintaining the properties like consistency, style compliance, reliability. Whereas the latter work is concerned about the evaluation of software architecture by metrics, which is applicable to software patterns too. In the work of Dr Zayaraz [7], he did use different available methods within his evaluation framework. For example he applied the rules and principles of the Common Software Measurement International Consortium (COSMIC) Full Function Points with some metrics to measure the basic interaction parameters for some characteristics such as coupling, cohesion, and complexity on different patterns (e.g. Pipes and Filters). Also, he did use Analytical Hierarchy Process (AHP) for comparisons between different pattern structures for specific quality attributes. Both approaches are a good step forward to building a concrete relationship between patterns and QAs based on scientific methods and measurement, not just an observation or experience of the pattern author. More research effort needs to be done to answer some of the questions that have been illustrated in Figures 1a and 1b. This paper attempts to highlight some important factors that impact pattern usability within the software engineering discipline, that are caused by conflicts between several pattern resources regarding relationships between patterns and their quality attributes (Patt-QAs).
A. Rationale Of The Investigation Approach

This investigation was necessary for two reasons: (1) to emphasize the problem concept, (2) to increase the proposed solution value to pattern users. This accomplished based on three processes shown in Figure 2.

The first process is, to highlight the differences between definitions, terminologies and categorisation as factors that challenge identifying relationships between software patterns and QAs (Patt-QAs). Seven analysis steps have been carried out to satisfy the first process as described in Table I.

Second process is, to discuss some survey questionnaire results, which support the existence of the problems described in the first process. Also, it supports our proposed solution to build a database of the relationships between patterns and QAs. Thirdly are, generating a metrics suite designed to express the investigation undertaken for six credible and definitive sources of patterns with respect to their characteristics. The software patterns sources included in this study are:

1. [1] – the Gang of Four (GoF) book,
2. [2] – POSA-V1,
3. [8] – POSA-V2,
4. [9] – POSA-V3,
5. [10] – Software Engineering Institute – Software Architecture in Practice, and

The Selection of these sources is based on the authors' preliminary research, also by supportive respondent's answers to a questionnaire done in 2012, (by the researchers). Almost half of the respondents identified GoF and POSA books as their reliable, popular and known pattern references. While the [10] and [11] included in this study as important part that tackle architectural and Security patterns, which is valuable to the research main goal.

The rest of the paper is organized as follows: Section 2 discusses the problems associated with differing pattern definitions, terminologies and categorisation, then briefly argues how QA definitions, terminologies, and categorisation cause problems in identifying their relationship with software patterns. Section 3 introduces an example that supports our claims in Section 2. Section 4 lists issues arising from variation on both domain software patterns and QAs. Finally, in section 5. We introduce some important findings from our survey that supports this investigation and our proposed solution, then summarise the database information and structure in section 6. Followed by the conclusions Section 7.

II. PATTERNS AND QUALITY ATTRIBUTES REFINEMENT

To create or describe a pattern we should understand the concept of pattern and follow rules or constraints to document them in the right way. To assess patterns against QAs, we should do the same to the QAs concept. The rest of this section lays out the problems that existed within the concept and rules of creating and documenting patterns within software engineering, that have a direct impact on their utilization and evaluation. Also, this section presents justifications as to why we built a (Patt-QAs) relationships database, and some of the challenges that have been faced during this process.

A. Problems Discovered Within Current Pattern Definitions And Terminologies

Numerous pattern definitions are being suggested for varying contexts. It is therefore difficult to define patterns in commonly acceptable terms. However, it seems...
sufficient to say that a pattern is essentially the solution to a problem within a particular domain which can be applied to help resolve similar problems in different contexts within the same domain. The definition of ‘context’ has evolved over time, for the purpose of this paper/study we believe that Dey’s definition is the most appropriate and is probably the most widely accepted.

Dey’s defines the ‘context’ as «any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between (for example) a user and an application, including the user and the applications», [12]. The definition of a pattern as described by GoF is «a solution to a problem in a context». This definition, however, was unacceptable to Dick Gabriel [13], who believed that it failed to illustrate the significance of the concept, and may even cause misinterpretation amongst software professionals. Gabriel also believed that many of the existing pattern definitions were indistinct and did not accurately express the implications patterns have. He therefore proposed a new definition, amending an early version by [14]: «Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves». Likewise, [2], [15], [16], and Gabriel[13], each one have his own pattern definition.

Most of the definitions above share common key points with a few variations. Some are more elaborate than others or include some further important aspects such as forces. Defining the forces that drives and constraints the most appropriate solution to a problem in the form of a pattern is an important step during pattern creation [14].

### TABLE I DESCRIPTIONS OF THE 7 ANALYSIS STEPS FOR THE TARGETED RESOURCES

<table>
<thead>
<tr>
<th>Process</th>
<th>Investigation Steps</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pattern Resources Selection</td>
<td>Identifying the most widely and reliable resources within the field of software patterns through concrete literature review, which become the targeted resources for this investigation study.</td>
</tr>
<tr>
<td>2</td>
<td>Pattern Categorization Approach</td>
<td>Study and compare all categorization approaches within the selected resources.</td>
</tr>
<tr>
<td>3</td>
<td>Quality Descriptions</td>
<td>Selection Identifying one of the best-standardized practices in the field for defining and categorizing the quality attributes through a literature review. Then we use the selected approach for identifying the relationship between patterns and QAs. Also, we use it for comparisons between different quality attributes categorization schema within the targeted resources.</td>
</tr>
<tr>
<td>4</td>
<td>Creation of the Relationship Matrices</td>
<td>Based on the pattern descriptions within the targeted resources, and the description of QAs by the selected approach, we built relationship matrix for each resource and a common matrix for all of the resources that identified the relationship between patterns and QAs.</td>
</tr>
<tr>
<td>5</td>
<td>Creation of the Quality Attributes Categorization Tables</td>
<td>Based on the information collected from steps 1–5, we created comparisons tables for the QAs classifications, between selected QAs approach and others within the targeted resources.</td>
</tr>
<tr>
<td>6</td>
<td>Conflicts and Issues</td>
<td>Based on the investigation steps 1–6, we have identified any relationship conflicts and issues within the descriptions of patterns on targeted resources.</td>
</tr>
</tbody>
</table>

Furthermore, having different terminologies and names in real life to explain the same thing, often due to differences in cultural factors or language, is acceptable. However, this is improper in the context of software patterns, as it leads to confusion. It is therefore considered as an absence of standardization, which can cause major challenges, [17]. Therefore this research aims to minimize some of these challenges by explaining the problem area and introducing the (Patt-QAs) database with its benefits and features, (see Section 6).

Terminologies shown in the Figure 3 are being used within the current literature. For example, the Architectural-Styles termed by [18] and [10], Architectural-Pattern by [2] and [19], [20] and [15] name it a Conceptual-Model,
and a Conceptual-Pattern by [16]. Many software developers use patterns in different stages of the software development lifecycle. We believe that the problem context persists, while the context of developments changes as described by Figure 3. More (redundant) terminology increases the challenge of pattern usability. It appears to some readers that the terms described in Figure 3 suggest different concepts. But are they?

The philosophy surrounding the conceptual or architectural ‘model, style, and pattern’ in the aforementioned terms attempts to convey a single idea through various explanations. All of which share the concept, components, restraints, and relationships that focus on a high abstraction level. However, the conceptual models should be explained further through detailed descriptions, in order to be able to move from an architectural context to a design context and so forth.

We believe that all terminologies shown in Figure 3 do have the same concept of pattern, with minor differences, to fit into the various development contexts. Also, less terminology surrounding pattern, and a concrete description of a common formal term, lead to better utilization and understanding of software patterns, which shall minimize the confusion in the midst of its users. So, many existing definitions and terms for the same concept (as illustrated above) was a challenging factor during this study. As a result, and based on this study, the relationship data code has been created over all patterns included in the selected resources. All patterns for all levels of the development life cycle are gathered in one place, with an indication of their names, definitions, and categories, to help developers to compare and find relevant information regarding the included patterns and their relations with QAs with little time.

The same discussion above also applies to the design phase as briefly discussed below:

Alexander defines design as «a process of synthesis, a process of putting together things, a process of combination», [14].

According to [2], design patterns depict frequently occurring arrangements of interacting components, thus helping to resolve design dilemmas in a given frame of reference. What this essentially suggests is that a pattern cannot be translated into code, but rather the pattern should be moulded in a way that it provides a solution to the problem.

Whilst, currently software developers can select a pattern as an available code artefact, alter it to match his/her problem context and finally convert the entire package into code. Nonetheless, we agree with [2], that patterns should be highly generic with textual explanations in addition to block and connector diagrams, in order to support higher reusability in multiple contexts and better understanding. However, the textual explanations and the block and connector diagrams should not be arbitrary. Also, should be applied within a common standardized procedure or a framework.

Various definitions (rules), of design patterns that convey the diversity of terminology and description can be noticed by comparing between the definitions of [1], [2], [21]–[23]

To conclude, the concept of a repetitive ‘structural’ pattern theme can be used for describing the architecture, design, and implementation, and what’s different then? Is the changed context. So, reducing pattern documentation conflicts, needs more research and standardized procedures, to helps increase the effective use of patterns.

Same concept been discussed earlier in architectural level within Figure 3 description.

B. Problems Discovered Within Current Pattern Categorisations

Coupled with the expansion of pattern diversity, there is a corresponding rise in the emphasis on the obligation to categorize patterns. To meet this end, a categorization outline is employed to organize the patterns as a collection so as to make them accessible for searching and storing by users. For the purpose of this section we have add POSA-V4 with other resources from section 1A).

The classification approaches for the investigated resources are:

- The first and the second volumes are based on two primary categories: ‘pattern’ and ‘problem’ categories. The pattern category is subdivided into 3 types in both volumes, while problem category is organized into 10 types in POSA-V1, and 4 types in POSA-V2.
- POSA-V3 were based on 3 primary categories within the domain of typical resource management lifecycle. These categories were resource acquisition, resource lifecycle and resource release.
- POSA-V4, the patterns were categorized on the basis of 13 technical topics and distributed systems.
- GoF team, however, used a different approach, classifying patterns based on purpose and scope. The ‘purpose’ has been further sub-classified into creation, structural, and behavioural categories, while ‘scope’ into categories of classes and objects.
- SEI book by [10] contains architectural styles that are categorized on the basis of respective subjects and relations. [10] describe thirteen different styles, of which the five primary styles are independent components, data flow, data-centre, virtual machine, and the call and return. The primary styles signify the relationships amongst the sub-styles and their respective topics.
- The book on security patterns by [11] comprises pattern categories bearing reference to enterprise and system levels within the security domain, and is related to engineering and operations activities at all levels.

Based on this study we found that the description of the technical topics (POSA-V4) are the same as «technical problems», which shares the same concept of the «problem category» that have been recognized in volumes 1 and 2. For example, the From Mud To Structure, have been described as a problem category in POSA-V1, and as a technical topic in POSA-V4.

From comparing the targeted resources mentioned above, it is clear that there is no common approach for categorising patterns. However, we believe that the ‘problem’ category as a concept, is shared between many
pattern books, although under a variety of names, for example, it is named ‘purpose’ in GoF book; ‘problem’ in POSA-V1 and V2 and ‘technical topics’ in POSA-V4, and as ‘main style or related subject’ in SEI.

Also, as an example of confusing categorisation schema used in these books is that of the Interpreter pattern, where GoF considers this pattern as a design (behavioural) pattern, but the SEI group consider it an architectural (virtual machine) style. So, what is the Interpreter pattern, and does this affect the reusability of this pattern? Can we use the same pattern, that explained by GoF in the context of a virtual machine, as explained by SEI group, or do we need to adjust it to fit the new context?

This lack of a common classification, particularly for scenarios that are technical, such as software patterns, can end up complicating things for users, researchers and readers. Therefore, when users seek appropriate patterns for resolving certain real-life issues, they are confronted with different guides and classifications for what are essentially the same patterns. Whilst, this can assist the users in employing the patterns in diverse contexts, it may also contribute towards making the reuse factor of patterns more complex, unmanageable, and less efficient. To assist with minimising such confusion, this study provides a database with information regarding 168 pattern (in-total) names and classification, helping developers compare and choose the most appropriate patterns for their problem domain.

C. The Variation Concept As A Problem within QAs

There are many different schools of thought regarding the management of QAs and how they can be addressed effectively such as, ISO, SEI, DoD STD, and IEEE, [24]. Hence, there are challenges that arise when quality has to be defined in the real world. This section tries to demonstrate in brief the difficulties that arose during this study from the QAs documentation variation viewpoint.

include all variants and relationships with quality attributes.

According to Mitra 2008 and reference therein pertaining to Juran and Gryna (1993), Crosby (1979), IEEE-1061, and ISO-9126, each have their own individual concept of quality. Doctor Ronald [25], argues that there are variations in QA definitions that are acknowledged by both the community and researchers involved. The presence of different concepts of quality amongst different people and communities illustrates that there are variations within the definitions for each QA that may share some characteristics and differ in others. However, small variation within QA definitions could increase the difficulties in defining and evaluating software patterns against them.

Likewise, the terminological variations concept persists with QA categorisations, same as the pattern categorisations issue discussed earlier. So, depending on the domain, people have designed different ways to classify QAs using different approaches. The needs for further research and study increased; however this will not be discussed in this paper. The focus here is to explore the differences in QA categorisations within our six sources and demonstrate the issues elucidated by these differences, which will be discussed in section 3 and 4. However, the relationships database included all the QAs definitions and categorisations for ISO 9126, because this a standards represents a broad agreement of QAs. Also, QAs definitions and categorisations for all targeted resources where applicable is included in the database, to help the users to make their comparisons between different approaches.

III. CONFLICT EXAMPLE - (PROXY PATTERN)

This example for illustrative purposes of the issues discussed in Section 2. It is a comparison of the Proxy pattern documentation approach, between the GoF and POSA-V1. This comparison shows some of the differences that we think lead to confusion and that minimize the utilisation of software patterns.

The definition of the Proxy pattern has similarities in both resources. While, POSA-V1 did elaborate further in their description. However, there are more differences within the Proxy pattern such as: (1) their instances or variants, (2) their primary and secondary categorisations, (3) their relationships with QAs. Figure 4 and Figure 5, visualize the above three differences.

The GoF divides Proxy patterns into 4 variants: remote, virtual, protection and smart reference as presented in Figure 4. Contrastingly, the POSA group divide the Proxy pattern into 7 variants, namely remote, protection, cache, synchronisation, counting, virtual, and firewall as seen in Figure 5. The common variants between both methods of classification are remote, virtual and protection. The important question being, which QAs are supported or hindered by those variants in both references.
Figure 4 shows that all GoF Proxy pattern variants support ‘lowering cost’ as a QA, and Virtual and Protection patterns supporting optimization and security respectively.

The POSA team on the other hand considered all Proxy pattern variants, including common ones such as Remote, Virtual and Protection patterns, to be supportive of usability, security, and performance. Unlike the GoF scheme, efficiency and lower-cost are supported only by the Virtual pattern. Whereas, efficiency is hindered by all other variants, as shown in POSA team approach, Figure 5. The above divergence in the categorizations schema and relationships between patterns with QAs increase confusion, making it harder to predict outcome quality when utilizing these patterns, as well as reducing pattern usability.

IV. ISSUES DISCOVERED BY THIS STUDY

- There are no specific definitions or categorisations of QAs that are presented by [1]. The approach taken instead focuses on the explanation of how patterns can be used to support claimed QAs. They used their own words and examples to explain QAs in the context of software patterns.
- ISO-9126, POSA Books, and SEI [10], defined QAs with various differences using various vocabularies. Although the concepts of their definitions are largely similar for each QA. However, they do varied in their sentence structuring, terminologies and how many features or constraints are included within their definitions. We believe, that any additional (features or constraints) added to any QA definition should be considered as a prerequisite that needs to be fulfilled, to achieve that QA with all its characteristics. As a result, the above variations in the QAs descriptions could have an impact on the overall evaluation process for any system or structure (e.g. patterns), and cause a conflict between development’ teams if they use non-common descriptions for the intended requirements (e.g. QAs).
- ISO-9126, POSA Books, and SEI [10] present different QA categories. For example, ISO-9126 and POSA Books, each have ‘Reliability’ as one of their main categories, but they differ in their sub-categories as illustrated in Figure 6 which is clear then that we will experience differences when trying to satisfy or validate the ‘Reliability’ QA using both approaches. For more information, see the QAs categorisation table in the database.

- One of the biggest causes of confusion and difficulty in traceability is the use of different names for the same patterns or one name for different patterns. For example, GoF team explained Adapter and Decorator patterns as two different patterns, which they are. However, both have been identified as Wrapper pattern. It is neither logical nor user-friendly for the same pattern to have different names or different patterns have the same name, making it hard to identify, trace and apply. It is understandable to have a variety of names if the pattern has individual instances or variants, such as the Proxy variants example discussed earlier. There are other examples of this «documentation problem» where the same pattern has various titles: Publisher-subscriber, Observer and Dependents are all different names for the same pattern. Indeed there are 8 different names described by [11] for Check-Point pattern alone, which are (Policy Definition Point (PDP), Policy Enforcement Point (PEP), Access Verification, Holding off hackers, Validation and Penalization, Make the Punishment fit the Crime, Validation Screen, Pluggable Authentication). However, GoF and POSA books have provided something as a solution to this problem, by introducing «Also Known As» section. Other resources such as [3] and [10], however do not acknowledge alternative names in their work.

Some resources include the same patterns with the same names and definition, but with different QA relationships. For example, in POSA-V1, the Piping and Filtering pattern supports Testability and Exchangeability, whereas SEI book lists it as supporting Maintainability and Usability. Questions therefore arise as to which QAs the pattern truly supports, and how these different conclusions have been reached. Not forgetting that QA relationships seem arbitrary, and the answer most probably lies with the differing experience and observations of the pattern authors, or because there is still a lack of proper methodology to capturing and documenting patterns, as we believe.

<table>
<thead>
<tr>
<th>TABLE II: METHODS SELECTED DURING THIS ANALYSIS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual analysis methods</td>
</tr>
<tr>
<td>Several types of graphs (e.g. bar chart, pie chart), frequency tables, descriptive statistics, a nonparametric Chi-square, and numerical measurement for the (Likert) type questions.</td>
</tr>
</tbody>
</table>

Copyright to IAUJSSET www.iarjset.com
Using expert knowledge regarding recurring problems to provide feasible solutions to the community relies on good standardized documentation, as recommended by [17], that standardisation helps decrease the challenges facing software development, preventing user confusion. To follow Garlan advise, we used the ISO-9126 model as the reference from which to build the relationship matrices between patterns and QAs, using the information described in all 6 resources studied.

V. PATTERN QUESTIONNAIRE AND ANALYSIS

In this section the researchers report the results of a survey designed to establish the reasons affecting the utilization of software patterns. A (secondary) goal of the survey was to obtain the agreement of survey respondents to some proposed solutions that could help developers with understanding better the effective use of patterns during the process of selecting and deploying them. A high level of confidentiality was applied during gathering and analyzing responds. The following sections outline the process and methods used in the analysis of the responses to the questionnaire. Section 5C shows an important portion of the questionnaire that is related to the scope of this paper. During this analysis the Statistical Package for the Social Sciences (SPSS) tool, was used. This explanation is to facilitate tables and figures notations.

A. ANALYSIS PROCEDURE

The survey was divided into three different sections as follows:

The first section focused on gathering information regarding respondents personal expertise. The second section centred on determining the reasons that affect the usability factor of software patterns during development processes. The last section was aimed at discovering issues that are related to current software patterns documentation, and also, to obtain the respondents’ agreement regarding

| Q1 | Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field. |
| Q2 | Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess. |
| Q3 | Studying relationships between patterns and quality attributes |

some proposed solutions by the researches. The analysis procedure was carried out in two steps as follows:

1. One-dimensional Analyses. Each question was analyzed and summarized in the form of graphs and tables where needed.
2. Multi-dimensional Analyses. In this step we analyzed more than one question together (matrix-cross-correlational), to see if there are any relationships or dependences between various factors. The selections of the questions were based on the overall objective of the investigation.

B. JUSTIFICATIONS OF THE METHODS USED DURING THIS ANALYSIS

Several techniques were used to carry out this analysis, the selection of the methods based on best technique that suit the type of questions, such as questions with ordinal scale, t-tests were employed, and for dichotomous variables a pie charts were used etc. However, due to the paper limitations we briefly named the methods that been used for each category, see Table II.

C. RELATED ANALYSIS

In this section we will show the statistical results of the general agreement amongst the questionnaire respondent towards the four mentioned statements that shown in Table \ref{tab:4Q}. These questions was proposed as a solutions to some of the issues discussed in sections 2, 3, and 4 each of the statements responses were of Likert scale, variables are of ordinal scale, so numerical measurements are meaningful. Assigning 1 to Strongly Disagree to 5 to Strongly Agree, the neutral option was assigned to 3 as it is value. So, one interesting matter is to see whether there any tendency to «Strongly Agree» or «Strongly Disagree»s. A one sample right tail t-test will be useful to see the general agreement of the respondents (see Table IV).

So, our hypothesis will be based on the neutral selection (Neutral value = 3), as follows:

Null hypothesis, \(H_0: \mu \leq 3\)

Alternative hypothesis, \(H_1: \mu > 3\)

where, \(\mu\) is the mean score of each of the statements.

Statistical analysis results are:

The overlapping (95 percent CI) error bars on the (Agree=4, option) indicated that most of the respondent agreed with all four statements as illustrated by Figure 7.

Table III: The 4 Questions – That Supports this Study.
Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.

Aslo, to investigate the respondents agreement significance with 95 percent confidence interval towards the four mentioned statements, one sample (right tailed) t-tests were performed. The test is significant for all of the statements as described in Table IV.

To sum up, this paper presents the work that satisfied part of the respondents’ wishes in Q17 and Q19 (see Table III) and to contributes to software patterns community, by identifying the relationships between some existing software patterns and QAs. Also, by developing a database to represent this information in easy way for the users. More research needed to provide solutions to the statements presented in Q18 and Q20 above, (see Table III).

VI. BRIEF DESCRIPTION OF DATABASE OF PATTERNS VS QUALITY ATTRIBUTES RELATIONSHIPS

It is recognized the importance of software patterns and QAs relationships to the software development processes. Investigating, and analyzing of these relations were carried out to help users to locate their desired relationship in short time and easy way through the developed database. There are several tabs, each one have many services. We recommend users to start with the overview tab to understand the overall structure of the database, and to facilitate their navigation process.

In total, we categorised 168 patterns and identified/systematised the known relationships between 120 patterns and 50 QAs within our database. Our database contains these relationships as well as other features such as search functions that can be used to easily find any patterns, conflict relationship or QA. Users can therefore explore each reference included in this study in an individual matrix, or view the pattern categorisation table for an individual resource. Each pattern has a description table consisting of definitions, alternative...
names, comments and relationships. A contrast tables of QAs classifications between POSA, SEI and ISO-9126 is also included.

In the future, the description table will needs further updating in order to enhance knowledge about patterns and QAs. Furthermore, they will also include forces, scenarios, quality tactics and quality metrics, as well as other information deemed essential for comprehensive knowledge about software patterns and their QAs relationships. In addition, the database built to be easier to explore as well as navigate through the user-friendly interface and menu. Users are therefore in a position to create, delete or even modify any relation. This database means that all the information on this subject is gathered into one place, providing summaries for numerous resources. The importance of the database comes from its ability to effectively save users time and effort, especially those who are concerned with finding a brief summary about particular patterns. To conclude this section, developing the database was very hard and time consuming, due to all processes involved from investigating to representing the information included. As a result, the database application was produced in such manner that it will be practical to other researchers and analyst. The database could be navigated with a proper access authorization through the researchers.

VII. CONCLUSION
This investigation of the relationship between QAs and software patterns has highlighted two main issues. Firstly, there are differences between pattern documentation within the current literature, which may because of different factors such as, authors experience and the maturity of the patterns in the field of software engineering. Secondly, there’s isn’t concrete approach or process to be followed for describing the relationship between patterns and quality attributes, or for categorizing them in a more sensible formal/verifiable way. Both points above have lead to the existence of conflict relationships between patterns and QAs, which decreases the utilization of patterns by users. Our major research objective is to aid software engineering community to see and help overcome the pattern documentation problem that we have identified. Also, to help patterns users to build better software by selecting patterns without ignoring their quality attributes, through visualizing this relationships within presented database, which been identified by several credible resources in the field. We believe that mining software patterns and pointing to any issues within their descriptions is an important step to improve pattern documentation, which already have a major affect on distilling and documenting software artifacts during software development lifecycle as discussed in Sections 1, 2 and 3.

REFERENCES
[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. USA: Addison-Wesley, 1995.

BIographies
Hassan Almari (1971) received his MSE and ME (hons) degrees in software engineering and engineering from the Australian National University - ANU, Canberra, in 2009 and 2010, respectively, and he is currently working toward the PhD degree in software engineering field. He have over 17 years of experience in computer and operational department in RSADF, he is Lieutenant colonel in the RSADF until now. His work experience includes developing and maintaining C3 and C4I systems. In 1998 he worked for 6 years in the PHI project, to
integrate several weapons into one command and control center (C3) with RSADF and Colsa Corporation. His main research interest and current work focused on architectural level, and it is modeling and evaluation techniques.

Clive Boughton (1956) possesses a PhD in Molecular Physics from the Australian National University (ANU). Clive is a professional who possesses over thirty years of practical experience in varying roles as scientist, engineer, software engineer, consultant, academic, and project and company manager. His collective experiences have given him the opportunity to observe and contribute to commerce and defence industries using contemporary techniques, languages and management methods. His extensive industry experience led to him attaining tenure in Computer Science at ANU, where he finalised a four year degree in Software Engineering, and set up a Masters in Software Engineering. His teaching and research interests include requirements elicitation and analysis, project management, quality management and systems/software modelling and architecture. He has supervised several PhD and Masters students surrounding topics in Software Engineering.
Questionnaire Report on Matters Relating to Software Patterns

Hassan Almari
Australian National University,
CSIT Building 108 North road
Canberra, ACT 0200, Australia
email: hassan.almari@anu.edu.au

Clive Boughton
Australian National University,
CSIT Building 108, North road
Canberra, ACT 0200 Australia
Chairman and Director of Software Improvements
GPO Box 1928, Canberra, ACT 2601, Australia
email: clive.boughton@anu.edu.au

Abstract—The development of software patterns is aimed at providing a reliable and reusable framework for resolving similar problems within distinct contexts. For this objective to be accomplished competently, the patterns have to be documented effectively to enable users to comprehend their concepts, and encourage their frequent use. Included within the documentation of patterns must be explicit explanation of their relationship with the known quality attributes (QAs) that they support or hinder, in order to fully satisfy the implementation of the requirements of stakeholders.

The variation in description of patterns in contemporary literature makes the explanation of the relationship given above complex and difficult to follow. The eventual result is that developers are deterred from utilizing patterns or that they ignore quality attributes. Either of these scenarios may render the process of development significantly expensive in terms of time taken, and/or attaining required system quality. This paper tries to address the aforementioned problem by identifying the factors that could facilitate or hinder the utilization of software patterns through an online questionnaire targeting participants with relevant experience in the field of software/systems development.

Index Terms—Software Patterns, Quality Attributes, Utilization, Evaluation.

I. INTRODUCTION

Various studies have attempted to detect, examine, assess, and describe the relationship between styles/patterns and quality attributes (QAs). Every study takes a unique focus with a scope within a particular QA and field, such as [1], [2], and [3]. Nevertheless, hardly any such studies have pointed out the relationship between architecture (including patterns) and quality attributes within a broad scheme that could be used for the majority of the (known) quality attributes.

The Research School of Computer Science in Australian National University conducted research aimed at determining the factors that influence the utilization of patterns in software development and some of the challenges faced by developers while selecting and deploying software patterns. A (secondary) goal of the research was to identify the factors that are likely to (as opposed to prevent) aid developers with the effective use of patterns during the process of software development.

The research began by drafting a questionnaire the focus of which was to uncover how important it is to developers to use software patterns with more efficiency and with quality attributes in mind. A high level of confidentiality was exercised in treatment of the responses to the questionnaire and this paper reports the analyses of the findings surrounding those responses. The following section presents an overview of the procedure used in the analysis of the responses to the questionnaire. The Statistical Package for the Social Sciences (SPSS) tool, was used during this analysis. This clarification is to facilitate tables and figures notations.

II. ANALYSIS PROCEDURE

The questionnaire was divided into three different sections as illustrated in Figure 1.

![Fig. 1. The three sections of the questionnaire.](image_url)

The first section was intended to gather information about personal expertise of the different respondents. The second section was aimed at discovering the different factors that affect the use of software patterns during software development, while the third and last section was aimed at finding out how documentation of software patterns could affect the utilization of software patterns during software development.

The analysis of the responses was carried out in two steps as follows:

1) Individual analysis, treated in one dimensional. The responses to each question were summarized and presented in graphs and tables where necessary.

2) Various sets of questions were analyzed collectively. The choice of the questions was based on the general objective of the research – to find the strength of dependences/relationships between various results. This matrix (cross-correlational) form of analysis meant that two or more questions were analyzed together. This was carried out in two, three, and four dimensions respectively depending on the number of questions involved.
III. RATIONALE OF THE ANALYSIS METHOD SELECTED

Various methods were used to analyse the questionnaire responses as described below.

A. Individual analysis methods

For the individual analyse all responses with Yes/No were presented on a bar graph to show the total count for every group. A frequency table was also produced showing the response count and percentage for each group.

Descriptive statistics were used to analyze questions with categorical responses but without numerical values. The results were presented using a frequency table, pie chart and a non-parametric Chi-square test to discover any general tendency of respondents to certain questions. Questions with dichotomous variables were displayed on a pie chart.

‘Likert’ type questions provided numerical measurement responses where the middle number was treated as the Neutral value.

B. Multi-dimensional analysis methods

Two dimensional questions were also analyzed in various ways ranging from a one sample t-test to a cross tabulation with Chi-squares, bar charts, scatter plots and descriptive statistics among others. Since most of the questions were of nominal scale and only four were in the ordinal scale, cross tabulation was considered the most important method. For those with ordinal scale, t-tests were used.

The analysis of results using more than two dimensions was seen likely to produce insignificant results as illustrated by analyzing of different sets of questions such as (Q1, Q2 and each question from Q17 to 20, as well as Q1, Q3 and each question from Q17 to Q20) later in the report. Hence no 3-dimensional or 4-dimensional analyses were done in this study.

IV. STUDY FINDINGS

This section reports the findings of the various analyses carried out on the questionnaire responses, using the categories of analysis described above.

A. One dimensional Analyses

According to the responses to the first question (Q1), the largest percentage of respondents specialized in coding, representing 67 percent of all the respondents to the question and 20.7 percent of the total number of respondents. Design, Project Management and Recruitment follow in that order. Most of these experts have gained most of their general software development experience in academia, followed by industry and government respectively (Q2), as shown in Table I.

| TABLE I | SECTORS WHERE RESPONDENTS GAINED MOST EXPERIENCE |
|-----------------|-----------------|-----------------|
| Academia | Industry | Government |
| 40.1 | 34.6 | 25 |

However, at 5 percent significance level of the Chi-square test the differences are insignificant, leading to the conclusion that no sector is dominating. The results are tabulated as shown in Table II.

The study results also indicate that most of the developers, (61.5 percent) have 5–10 years of experience and only 15.4 percent have more than 15 years experience. According to the Chi-square test, these results are significant at the 1 percent level. This fact should not affect the study since even those developers with only 5 years experience had enough exposure to at least know about software patterns, ignoring other factors such as lack of interest that may affect this, as illustrated in Figure 2.

![Fig. 2. Respondents level of experience in software development.](image)

Most of the developers that responded were aware of software styles/patterns. This is an indicator that the questions pertaining to software patterns in the questionnaire could be answered from a knowledge point of view. However, the minority that were not aware of software patterns may still pose a problem regarding the knowledge and utilization of software patterns - largely because if we consider unaware participants educational time in the software field (typically three to four years) plus the survey minimum experience of five years, then they’ve had a total of about nine years experience in the software field without knowing what a software pattern is! (see Figure 3).

Out of the respondents, only 8 percent used software patterns in their work frequently, while 30 percent of the developers never used software patterns and 16 percent used them infrequently. The Chi-Square test points out that the difference in the frequency of use of patterns is significant at 1% level (Chi-square = 22.42, p value < 0.01). This indicates that there could be factors that affect the frequency of use of software patterns in the developers’ work.
As identified by the respondents, some of the main factors that discourage the utilization of software patterns include very little teaching of the topic in learning institutions (as identified by 46 percent of the respondents) and the difficulty to integrate software patterns with other components or existing systems (as identified by 41 percent). This indicates that there is still a lot to be done to increase the patterns usability factor, which will in turn increase the utilization of software patterns.

Nonetheless, 40.5 percent of the respondents identified that one of the main factors that encourage the utilization of patterns in software development is ease of implementation. The other factors which are also significantly important include the ease to find the right patterns to solve a problem encountered (35.1 percent), clear identification of quality attributes possessed by software patterns (35.1 percent) and proper and thorough documentation in references (32.4 percent). All these factors are therefore important in encouraging the utilization of software patterns.

Keen consideration of the quality of software patterns before utilization is key during the whole development process. It is therefore important for the developer to consider the quality of patterns before utilizing them. Although the majority of the respondents considered the quality of software patterns before use, a significant portion of the developers did not; Figure 4 illustrates the result. This latter behaviour could have a major impact on the development process. If developers were to combine several patterns in system development, understanding the pattern qualities is important in order to build a system with concrete qualities.

A majority (73 percent) of the respondents who had used software patterns in the past were willing to continue using them in future. Some of the comments entered by those who were not willing to continue using software patterns are key to revealing some of the drawbacks encountered in the utilization of software patterns. One major drawback identified from this was poor documentation and suitability to the problem area.

Regarding standardization of existed software pattern references (Q11), there was an almost equally split opinion as to whether their descriptions present a standardized way for creating and explaining software patterns. A majority however indicated that software pattern references do not follow a standardized procedure of documenting patterns, as demonstrated in Figure 5.

In relation to the identification and clear statement(s) of relationships between patterns and quality attributes in the software pattern references, only 24 percent of the respondents were in agreement that relationships have been clearly identified and stated, while 17 percent disagreed. Fifty eight percent were not sure, maybe because of the lack of description within the references they used or because of their (low) level of knowledge concerning references.

In relation to the analysis above, only 18 percent further agreed that the relationships had been proven scientifically or otherwise, while 25 percent disagreed. The rest (57 percent) were not sure about this. We believe that there are some attempts within software engineering domain to prove the above relationship scientifically. However, these efforts remain immature and we support the ‘disagreement’ group in this matter.

Eleven percent of the respondents also identified conflicting views among the references regarding patterns and quality attributes. According to some of the respondents, the cause of inconsistencies may be due to the differing views among authors about when the patterns will be used. These inconsistencies can make the process of software development very ineffective in terms of cost and time. This identifies a need for software engineering professionals to apply more effort to systemize the documentation of software patterns to eliminate such inconsistencies.

Possibly due to the disadvantages arising from the inconsistencies above, almost two thirds of respondents did not support having different names for similar patterns, terming it to be
against quality attributes. This model should be easy to use.

Within responses to the last four questions (the Likert type questions) there was strong agreement about the importance of software engineers and developers acquiring knowledge on patterns; which could be improved by the following contributions:

a) the identification of the relationship between software patterns and quality attributes based on the current reliable software pattern references; b) in order to identify the relationships in (a) above, it is important to identify standard quality attribute definitions within the current pattern references; c) it is important to create a database for storing the relationships identified based on (a) and (b) above; and finally, d) develop and provide an evaluation model to assess software patterns against quality attributes. This model should be easy to use.

However, a third supported the idea, giving the reasons that it would be necessary, especially if the patterns are similar but not identical, and in case of different contexts of application. It could therefore be important to have variant names for patterns used to solve different issues. However, to avoid confusion it is suggested that the variants should be extensions to the main pattern name for ease of identification, such as Proxy-Remote pattern or Proxy-Virtual pattern.

Sixty-four percent of respondents support the idea of having standardized documentation practice for software patterns. Only one respondent was against this but with no sufficient reasons, as can be seen in Figure 7.

Within responses to the last four questions (the Likert type questions) there was strong agreement about the importance of software engineers and developers acquiring knowledge on patterns; which could be improved by the following contributions:

a) the identification of the relationship between software patterns and quality attributes based on the current reliable software pattern references; b) in order to identify the relationships in (a) above, it is important to identify standard quality attribute definitions within the current pattern references; c) it is important to create a database for storing the relationships identified based on (a) and (b) above; and finally, d) develop and provide an evaluation model to assess software patterns against quality attributes. This model should be easy to use.

Do not support different names

Do support different names

Fig. 6. Proportion of respondents who do/do not support different names for similar patterns.

However, a third supported the idea, giving the reasons that it would be necessary, especially if the patterns are similar but not identical, and in case of different contexts of application. It could therefore be important to have variant names for patterns used to solve different issues. However, to avoid confusion it is suggested that the variants should be extensions to the main pattern name for ease of identification, such as Proxy-Remote pattern or Proxy-Virtual pattern.

Sixty-four percent of respondents support the idea of having standardized documentation practice for software patterns. Only one respondent was against this but with no sufficient reasons, as can be seen in Figure 7.

Within responses to the last four questions (the Likert type questions) there was strong agreement about the importance of software engineers and developers acquiring knowledge on patterns; which could be improved by the following contributions:

a) the identification of the relationship between software patterns and quality attributes based on the current reliable software pattern references; b) in order to identify the relationships in (a) above, it is important to identify standard quality attribute definitions within the current pattern references; c) it is important to create a database for storing the relationships identified based on (a) and (b) above; and finally, d) develop and provide an evaluation model to assess software patterns against quality attributes. This model should be easy to use.

However, a third supported the idea, giving the reasons that it would be necessary, especially if the patterns are similar but not identical, and in case of different contexts of application. It could therefore be important to have variant names for patterns used to solve different issues. However, to avoid confusion it is suggested that the variants should be extensions to the main pattern name for ease of identification, such as Proxy-Remote pattern or Proxy-Virtual pattern.

Sixty-four percent of respondents support the idea of having standardized documentation practice for software patterns. Only one respondent was against this but with no sufficient reasons, as can be seen in Figure 7.

Within responses to the last four questions (the Likert type questions) there was strong agreement about the importance of software engineers and developers acquiring knowledge on patterns; which could be improved by the following contributions:

a) the identification of the relationship between software patterns and quality attributes based on the current reliable software pattern references; b) in order to identify the relationships in (a) above, it is important to identify standard quality attribute definitions within the current pattern references; c) it is important to create a database for storing the relationships identified based on (a) and (b) above; and finally, d) develop and provide an evaluation model to assess software patterns against quality attributes. This model should be easy to use.

However, a third supported the idea, giving the reasons that it would be necessary, especially if the patterns are similar but not identical, and in case of different contexts of application. It could therefore be important to have variant names for patterns used to solve different issues. However, to avoid confusion it is suggested that the variants should be extensions to the main pattern name for ease of identification, such as Proxy-Remote pattern or Proxy-Virtual pattern.

Sixty-four percent of respondents support the idea of having standardized documentation practice for software patterns. Only one respondent was against this but with no sufficient reasons, as can be seen in Figure 7.

Within responses to the last four questions (the Likert type questions) there was strong agreement about the importance of software engineers and developers acquiring knowledge on patterns; which could be improved by the following contributions:

a) the identification of the relationship between software patterns and quality attributes based on the current reliable software pattern references; b) in order to identify the relationships in (a) above, it is important to identify standard quality attribute definitions within the current pattern references; c) it is important to create a database for storing the relationships identified based on (a) and (b) above; and finally, d) develop and provide an evaluation model to assess software patterns against quality attributes. This model should be easy to use.

Two dimensional analysis was applied in order to find out whether/how different attributes depended on each other. First of all, the relationship between how frequently software patterns were used by developers and their field of expertise in software development. It was concluded that the two are connected, and the analysis showed that software styles/patterns were utilized mostly by those whose expertise lies in architecture and design. This makes sense since in some fields, software patterns may not yet exist. The reason could be lack of clear documentation in some software domains.

The analysis of the relationship between the respondents’ work place (sectors) and how they used software patterns/styles tended to indicate that academics used software patterns less frequently than those in government and industry. However, this relationship was not very significant, according to the Chi-square test. The study also shows that developers from academia and industry consider the quality attributes of software patterns before utilization compared to those in government. However, this relationship was not clearly defined either according to the Chi-square test.

In analyzing the three statements about important knowledge needed by software developers (as analyzed earlier), the matrix analysis shows that those with experience in government agree strongest, followed by those in industry and finally those in academia. Those in government and industry agreed more with the survey statements that suggested solutions to the problem that has been identified in this paper abstract. This agreement makes reasonable sense in pointing to the problem and providing a solution for it. Those in academia on the other hand are more research oriented and thus tend to focus on identifying the problem, while those in government and industry are keen to provide a practical solution to problems.

A positive relationship was identified between the years of experience for developers and their awareness about software patterns. The more experienced the respondents were, the more likely they were to be aware of software patterns. However, the Chi-square test indicated that the dependence on awareness about software patterns on the year of experience in software development was insignificant.

Although preliminary matrix analysis tended to show that most of the respondents who used software patterns in development had less than 15 years of experience, a hypothesized Chi-square test ruled out the dependence of usage of software patterns on the years of experience. In the same way, it was noticed that the identification of conflicting views in the documentation of software patterns was not dependent on the years of experience in software development.

Generally, the number of years of experience in software development did not seem to affect the respondents’ decision to be in favor of different names for the same software patterns, support for standardized documentation of software patterns references, or the agreement with the statements about the important knowledge that needs to be possessed by software engineers and developers.
To analyse the dependency between (software pattern) awareness by developers’ (Q4) and if the relationship between software patterns and quality attributes have been proven scientifically (Q13), this study found no bearing between the two because all of the respondents who responded to that question were aware of software patterns. In a similar way, the developers awareness did not seem to affect the respondents’ opinions about the four statements (Q17 to Q20) regarding the knowledge that is key for improving software developers.

The next part of the study was to analyse the effect of how often the developers used software patterns. There is no direct relationship between the frequency of use of software patterns and the respondents’ opinion on the main factors that discourage the utilization of software patterns. Similarly, the frequency of use did not seem to be related to the respondents’ opinion on which are the main factors that encourage the utilization of software patterns. This leads to the conclusion that there is no significant impact of the main factors that either encourage or discourage the utilization of software patterns or how frequently the developers use software patterns.

There is however a clear difference among the frequency of use of software patterns, for those developers who consider the quality attributes of software patterns before utilizing them. This is a clear indication that developers who consider the quality attributes of software patterns before use, tend to use software patterns more frequently compared to those who do not consider the QAs. This makes sense because as developers use software patterns more and more, they also develop the need to know their characteristics.

Another significant relationship was identified between developers who have used software patterns in the past and will continue using software patterns in the future and the current frequency of use. It indicates that those who are willing to use, or have used software patterns before, use them more frequently. Further, analysis indicates that there is no significant relationship between the ability to identify conflicting views on software patterns and their frequency of use. This also was the case when different names for the same software patterns were supported by respondents, and when standard documentation of software patterns were supported. Finally, no relationship was identified between the criteria used to sample the opinions of respondents on the four statements about what could improve software developers knowledge and their frequency of use of software patterns.

In evaluating the relationship of different responses to the developers’ opinion on the main factors that discourage the utilization of software patterns, there was seen to be no significant relationship between those opinions and the developers’ support for standard documentation. However, a significant variation was noticed with the responses to the four statements trying to create a solution to some of the existing issues in software patterns field. This variation supports the survey proposed solutions stated in the last four questions (the Likert type questions), which were discussed earlier. Similar findings were also noted in the case of the main factors that encourage software pattern utilization.

C. Three and Four Dimensional Analyses

As indicated earlier, 3 and 4-dimensional analysis of the responses in this questionnaire proved to be ineffective. This was established by trying to determine the relationship between the respondents area of expertise, where they gained most of their experience in software development, and their opinion on the four statements to aid in coming up with a solution to the issues that affect software patterns utilization. This 3-dimensional analysis provided a null difference among all the responses given. This was also the case when relating the respondents’ field of specialization, their total number of years of experience in the software development field, and the response to the four statements.

V. Conclusion and Recommendation

The analysis of the responses to a questionnaire concerning various matters relating to software patterns indicates that that the results can be relied upon to draw at least some conclusions about various issues affecting developers in their utilisation of software patterns. The trends noticed in the study are key to establishing courses of action, and areas where there is urgent need for a solution to some of the issues affecting the use of software patterns in software development such as, documenting, teaching, evaluating, and modelling patterns.

Although the utilization of software patterns in software development has been embraced by a significant number of developers, this study identifies some of the crucial factors that encourage/discourage developers to/not to use software patterns in software development. Finally, the study suggests some solutions that can be embraced to increase the proper utilization of software patterns in software development.

It is thus our recommendation that all the critical issues identified in this study be taken into consideration to ensure that more developers embrace the use of software patterns to ease their challenges with software development and to make them more efficient in the field. This study raises important questions about the impact of software styles/patterns on software architecture evaluation, especially for architects who are one of the most important developers and who use patterns most frequently. A continuation of the work is therefore necessary to gather the opinion of, and pertinent information from, developers concerning the relationship between styles/patterns and their affect on software architecture in a future questionnaire.

REFERENCES


Hassan Almari
Australian National University, CSIT Building 108 North road
Canberra, ACT 0200, Australia
email: hassan.almari@anu.edu.au

Clive Boughton
Australian National University, CSIT Building 108, North road
Canberra, ACT 0200 Australia
Chairman and Director of Software Improvements
GPO Box 1928, Canberra, ACT 2601, Australia
email: clive.boughton@anu.edu.au

Abstract—Evidence of the relationship between software architecture including its styles/patterns and quality attributes continues to grow, but largely remains an art rather than a science in terms of being able to predict a relevant architecture from (known) quality attributes.

The aim of this paper is to point out those aspects that influence utilization of software architecture artefacts and their evaluation by software developers, and is part of a continuing study the results of which are intended to aid professional software/system developers with their decisions surrounding choice of (concrete) software architectures.

The earlier part of the study produced an analysis report based on a survey titled “Questionnaire on matters relating to Software Patterns - 2012”. The survey was issued to software developers possessing more than 5 years experience, and produced significant results which, in turn, led to the need for this second survey. Ninety seven (97%) percent of the participants, from six different nations, answering the first questionnaire supported this further investigation.

Index Terms—Software Architecture, Quality Attributes, Evaluation, Utilization.

I. INTRODUCTION

The utilization of software/system architecture and associated evaluation methods in industry is influenced by a number of factors. Varied studies have tried to detect, examine, assess, and describe the link between software architecture (SA), and quality characteristics/attributes, such as work done by [1], [2], [3], and [4]. Every study takes a unique focus scoped to a particular quality attribute or a view of the architecture. In order to discover more about these factors that influence the relationship between SA and QAs, a questionnaire was produced to target experienced software/system developers to determine their views on the applicability of existing SA artefacts and their associated evaluation methods, as well as to identify the challenges to utilization of these artefacts and methods in the development lifecycle. This paper summarises the result from the second questionnaire on matters relating to Software Architecture Evaluation (SAE). The analysis of the responses to the survey was performed using the Statistical Package for the Social Sciences (SPSS) application. These facts have been mentioned in order to help simplify table and figure symbolizations. In this paper, the use of figures and tables have been minimized due to page limitation. Full analyses of the survey (311 pages) is available upon request, through contacting the authors via email.

II. ANALYSIS PROCEDURE

The questionnaire surrounding SAE comprised of 23 questions which were grouped into various sections such that each section focussed on a specific objective as described below.

The first section of the questionnaire consisted of five questions aimed at gaining better background knowledge of the participants. In the second section, a further five questions focussed on the factors that affect utilization of SA modeling techniques. This was incorporated in an effort to make the utilization of SA artefacts together with all its modelling descriptions a cost-effective and time-saving process. The final section focused mainly on SAE and its challenging factors.

In analyzing the responses both one dimensional analysis and two dimensional analysis were used. In one dimensional analysis, the responses to each of the 23 questions were analyzed individually and the results were illustrated using proper statistical methods, depending on the nature of each question. In order to draw more vivid conclusions, the dependencies between different SA utilization factors were also analysed. This was achieved by carrying out a two-dimensional analysis, where some of the questions were analyzed as pair combinations in order to satisfy some of intended objectives by researchers. The methods used for both one and two dimensional analysis are described in the next section.

III. RATIONALE OF THE ANALYSIS METHOD SELECTED

Several methods were used to analyse survey responses. The rationale for the methods used is given below:

A. Individual (One dimensional) analysis methods

For individual or one dimensional analysis, the method of analysis used depended on the type of questions being asked. Questions which required respondents to select a single response from a list of choices, such as Q1, were analysed by presenting all responses on a single bar chart showing the total count for each choice. The bar chart was derived...
from a frequency table which was used to represent the same information as percentages.

‘Descriptive statistics’ was the method used to analyze questions for which responses were divided into categories without numerical values. The results were presented using a frequency table to show the percentage for each respondent as well as the cumulative percentage of the responses. In addition to the frequency table, for questions which required the respondent to make a choice from numerical responses, a Chi Square Test was performed to show the distribution of the responses in each category in order to draw a more accurate conclusion.

Questions which involved dichotomous variables were presented on a pie chart for easy visualization. It was convenient to do this since for these types of questions, there wasnt any overlap of responses. Thus the total percentage from all categories of responses was 100%.

Responses to types of questions involving statements for which some level of the agreement was required were presented on a Likert scale to show the strength of agreement or disagreement. For these questions, the analysis was done by employing descriptive statistics. A one sample t-test was performed on the resultant mean for each of the questions in order to determine and confirm the significance of the concluding agreement level.

B. Multi-dimensional analysis methods

Multi-dimensional methods involving analysis responses to questions as a block, or in matrices, were employed to determine whether a relationship existed between the responses of the various combinations of questions chosen. The analysis methods included sample t-tests, cross tabulation, Chi square tests, Tukey’s HSD method, scatter plots, descriptive statistics and bar charts among others. Cross tabulation was preferred in most cases since most of the questions involved a cardinal scale, with only four questions lying on the ordinal scale. T-tests were used to analyze the latter category.

Three-dimensional and four-dimensional analyses were invalid, inappropriateness, and insignificance. The Chi-square test results for cross tabulations under three dimensional analysis are all invalid due to low cell (data) expected frequencies. Also, most of the t-tests and F-test results are all valid in statistical sense, though there were no significant results. So, they were excluded from this paper.

IV. STUDY FINDINGS

The findings of this study uncover many factors affecting the utilization of system/software architecture artefacts by system/software developers across various sectors (work place), during the development lifecycle. The findings are summarized in this section according to the analysis method used.

A. One dimensional Analyses

This research was conducted based on responses to a questionnaire distributed to participants from various fields of expertise, the majority of whom were designers, programmers, analysts, and architects among other fields as illustrated in Table I. A majority of these professionals have gained their expertise in academia, followed by industry, and Government as shown in Figure 1.

The Chi-Square test confirms the significance of the results at 5% level, Chi-square = 7.84, p-value < .05. The findings of the research can thus be considered credible due to the varied areas of expertise of the respondents to the questionnaire.

In order to increase the reliability of the results, it was also necessary to ensure that the participants had enough familiarity or experience in a (particular) field of software/system development. The experience of the majority of the participants ranged between 5 and 10 years (48 percent), with about 30 percent having an experience of more than 15 years and 22% having experience of 10 to 15 years. Thus, the years of experience considered sufficient to gain familiarity with the trends in software/system development.

More than 83 percent of all the participants indicated that they were aware of software/system architectural description/modelling languages, which increases the reliability of the responses given in the questionnaire, because they do have knowledge about the survey field, which help them to select a proper answers based on their experiences. At the same time, considering the experience levels, the finding raises a question about 17 percent of the participants in the software/system development industry who were not aware of software/system architectural description/modelling languages, which could be affected by different factors such as the lack of teaching SA in academic institutions.

Despite the majority awareness of software/system architectural description/modelling languages, an alarming 50 percent of the respondents either used models infrequently or did not use models at all. This justified the need for a further questionnaire to unearth the reasons that could be encouraging or discouraging the use of system/software architecture modelling languages.

The two major (out of 8) factors that encourage the utilization of models to describe software/system architecture as identified by the respondents include, “the ease to demonstrate the software/system concept and features” (65.2 percent), as well as its contribution in “making the designer/programmers’ job easier” (30.4 percent). On the other hand, one of the two main factors (out of 5) that discourage the utilization of modelling techniques to describe software/system architecture is
“the difficulty in integrating these models with other artefacts (e.g. Design models)”, which makes them standalone models rendering them less useful in the process of software/system development”. The other factor is “the lack of standardization between the existing architecture modelling techniques, notations and semantics”, among other factors.

In an attempt to determine the best language to use to describe software/system architecture in order to increase its usefulness to all stakeholders as well as the ease of qualitative and quantitative assessment, it was found out that a combination of semi-formal language and natural language used together would be preferred by the majority (52.2 valid percentage). Semi-formal language alone is the second most preferred (17.4 percent valid), while 13 percent prefer all three languages (formal, semi-formal and natural languages). The results above were expected, based on the researcher’s observations to several current projects documentations, within the Royal Saudi Air Force (RSAF). We think its a good combination to have semi-formal and natural language, due to the ease with which non-developer stakeholders can understand these languages as well as the ease of the evaluation processes.

There is a general agreement that developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DODAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture. On the other hand, there is neither agreement nor disagreement about whether “the usage of software style/pattern concepts & models during architecture development increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture evaluation or not”. This is confirmed by a one-sample t-test as described in Table II.

In an attempt to gather the level of awareness surrounding system/software architectural metrics, an overwhelming 85 percent denied having any awareness about any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models, (e.g. detecting attacks for security). This uncovers the great need to increase awareness about the existing tactics such as the effort done by [5], as well as the metrics such as the work done by [6] for security measurements, and [7] for applying metrics to assess software artefacts, and document them. Architectural evaluation methods suffer the same fate, with more than 89 percent of the respondents lacking awareness about any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics. The questionnaire therefore attempted to uncover the reasons that may be encouraging or discouraging this knowledge and/or use of software/system architectural evaluation methods.

According to the statistics gathered from the questionnaire, the two most important factors that could support quantitative evaluation for any software architecture (SA) include “availability of tools for describing and evaluating SA” (identified by 44 percent of the respondents) and “the use of standard language and architecture framework for describing SA” (identified by 40 percent of the respondents). On the other hand, two of the most important factors that hinder quantitative evaluation for any SA include the “formality level of SA description” as identified by more than half of the respondents, followed by “the language used for describing the SA” as identified by 40.5% of the respondents. These are the factors that need to concentrate on in order to improve the knowledge and use of system/software architectural artefacts and its evaluation methods that can produce qualitative and quantitative results.

The general agreement to the suggestions given by researchers in this questionnaire regarding important matters affecting architectural evaluation, and how we could aid resolving these issues, was measured as shown in Figure 2.

According to the analysis, there was a significant agreement to all the statements in Figure 2, except on the statement...
that “most of the existing software architecture evaluation methods, produce qualitative results”, to which the majority of the respondents remained neutral. However, the one-sample t-test on the agreement to that statement ($\mu > 3$) is statistically significant at 1% level, $t=2.46$, p-value $<.01$ implying a general agreement to the statement.

Finally, the agreement to the statements in Figure 3 were also tested using descriptive statistics.

From the figure it is clear that there was a general agreement to the first three statements presented. However, in regard to the last statement, about one third of the respondents (14 out of 43) either disagreed or strongly disagreed to the statement that stated “current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”. According to a continuing study done by the researchers since 2010, we believe that there is no existence of such a tool yet. However, the current technologies are becoming closer to achieve the above statement, and many gaps have been closed during the last decade by improving SA description languages and tools. Furthermore, the above results indicated that more research needs to be done to improve and contribute to SA evaluation filed.

In addition, a significant number of the respondents (18 out of 43) representing 42 percent remained neutral regarding the statement and only 26% (11 out of 43) either agreed or strongly agreed to this statement. The responses were mostly skewed around the neutral axis and a one-sample t-test on the agreement to the statement ($\mu > 3$) showed significance at 5% level with $t=-.80$, p-value $>.05$, indicating a general disagreement to the statement.

**B. Two dimension Analyses**

In order to establish the credibility of the responses as well as important trends in the responses given to the questionnaire, the questions were cross-tabulated in pairs to determine the association between each question in the pairs. This section summarizes the findings.

It was noticed that the general field of expertise regarding software development did not have any association with most of the other questions in the questionnaire. However, it was noticed that the most of the respondents whose field of expertise was Project management did not think that most of the existing software architecture evaluation methods, produce qualitative results, compared to other areas of expertise ($t=2.35$, p-value $<.05$). On the other hand, Architects, more than other fields of expertise, tend to think that “Architecture is design, but NOT all design is Architecture” ($t=2.20$, p-value $<.05$). Furthermore, there is significant inequality in means between Architects and experts in other fields regarding the statement that “Architecture is design, but NOT all design is Architecture” ($t=2.20$, p-value $<.05$). This shows that Architects embrace this statement more than developers in other areas of expertise. Whereas, those whose field of expertise is coding tend to disagree more than others to the same statement above, which is very important indication to the variation in understanding the architecture concept between both architects and programmers. As a result, architectural artefacts utilisation by programmers could be decrease due to their understanding of SA concept, which we believe its not correct.

Regarding software evaluation methods, experts in Project Management showed a much less significant result compared to those who have other expertise on the statement that “Most of the existing software architecture evaluation methods produce qualitative results”. While, designers showed a significantly less population mean than other experts to the same statement, ($t=-2.10$, p-value $<.05$). However, they also showed a greater population mean on the statement that “Reliable tools are important for developing/or evaluating software/system architectures” ($t=2.04$, p-value $<.05$).
On the other hand, experts in Coding showed a less significant population mean on the statement that “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures” (t = -1.87, p-value = .035 < .05). These results are a clear evidence of how the field of expertise of the respondents to the questionnaire affected their opinion regarding evaluation methods used in system/software architecture.

There are many reasons that could cause the above variations, such as developers SA background knowledge, the level of developers involvement in SA evaluation process etc. A similar variation in opinion was also noticed in matters relating to current technology as related to system/software evaluation methods. For example, the population mean for responses from experts in “Coding” to the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” is significantly less than that of other experts (t = -2.50, p-value = .008 < .01). This is also the case for the population mean of responses from experts in “Testing” towards the same statement (t = -1.79, p-value = .041 < .05). These relationships led us to appreciate the varied nature of needs in the software/system development industry according to the expertise, hence concluding that a careful consideration about the area of expertise while implementing any suggestions offered in this study is vital to make the implementation more impactful.

The number of years of experience of the respondents was seen to affect their opinion on the statement that “most of the existing software architecture evaluation methods, produce qualitative results”. Turkey’s HSD method was used to determine the variation in the responses and the association that exists between the two. The analysis uncovers a significant difference (Mean difference = 1.33, p-value = .011 < .05) between respondents with 5-10 years’ experience and those with over 25 years’ experience, with the agreement favouring the earlier group.

There is a higher rate of agreement among those who had awareness of modelling languages compared to those who didn’t, on the statements that “Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task”, (t = 1.9, p-value = .04 < .05) and “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”, (t = 2.1, p-value = .03 < .05). We agree with the respondents for their for the first statement, but not to the second one, due the same reasons discussed earlier a round Figure 3.

There seems to be a variety of relationships between the opinions of respondents regarding qualitative and quantitative system/software evaluation methods and how often these respondents use software/system models in the description of software/system architecture. For example, there is a significant statistical difference between group means corresponding to various categories of how often respondents use models to describe software/system architecture for the item “There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers”, (F = 2.7, p-value < .05). The group means for this item exhibit a difference among respondents who “Nearly always (>90%)” used models to describe software/system architecture during their work compared to those who “Never” used them at all, Mean Difference = 1.2, p-value < .01. Similarly, there is also a difference in the group mean for the item among those who “Nearly always (>90%)” used models to describe software/system architecture during their work and those who “Reasonably frequent (>15% and <50%)” used, Mean Difference = 0.7, p-value < .05.

Also, the test is significant for the item “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures”, F = 2.7, p-value < .05. The last difference is exhibited between those who “Nearly always (>90%)” used models to describe software/system architecture during their work are statistically different for the item “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures”.

These differences are an indicator of the effect of length of experience in software/system development on the responses received from respondents. This is the reason why we chose to use only respondents with above 5 years of experience in the questionnaire, in order to ensure that the opinions and experiences expressed are based on familiarity in the field of system/software development.

Another significant relationship noted was between the respondents who agree that use of models “makes the designers/programmers job much easier” and those who think that “Reliable tools are important for developing/or evaluating software/system architectures”. This positive relationship is a confirmation that the identification of the factors encouraging the use of models in software/system description is interdependent, depending on the general preferences of the respondents.

In a similar way, independent sample t-tests Chi-Square and ANOVA tests performed on various hypotheses revealed interesting relationships between various responses, reflecting the wide variety of preferences by the respondents. For example, an independent sample t-test the hypothesis (μ > 3) for the item “Reliable tools are important for developing/or evaluating software/system architectures ” showed significance at 5% level, t = 2.2, p-value < .05. This reveals that the population mean of the item “Reliable tools are important for developing/or evaluating software/system architectures” is significantly higher for the respondents who replied “Yes” to the item “It makes the designers/programmers job much easier”, compared to those who disagreed.

In a similar way, an independent sample t-test revealed a
direct relationship between the statements “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” and “Yes” responses to the item “Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier”. The test is significant at 5% level, t = 2.0, p-value = .049 < .05.

A positive result was similarly noticed between the population mean for the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” and “Yes” response to the item “The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible”. The test is significant at 5% level, t = 2.1, p-value < .05. Similarly, this was the case between the population mean for the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” and “Yes” responses to the item “Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools”. The test is significant at 5% level, t = 2.5, p-value < .05.

Once again, the above results point out clearly the relationship in opinion between matters relating to the use of current technology in software/system architecture and evaluation methods, which can aid in clear judgement when implementing the recommendations of the study in the software/system development industry.

Another significant relationship was noted via an independent sample t-test conducted to test the hypothesis \( \mu > 3 \) for the item “There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers” and those who admitted that it is “Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)”, \( t = -1.96, \) p-value = .028 < .05. This was the case as well for the item “Most of the existing software architecture evaluation methods, produce qualitative results” at 5% when using a two-tailed test, \( t = 1.94, \) p-value > .05.

In order to test the dependence between two categorical variables, a Chi-square test was used. For the relationship between the variables “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “Most of the existing software architecture evaluation methods, produce qualitative results”, the Chi-square test showed significance, \( \chi^2(16) = 60.10, \) p-value < .01.

The above are some of the relationships established between categorical variables, aiding further to deduce the trend in opinion amongst software/system developers in industry with the aim of improving their experience with system/software architecture in the future. In a similar way, other positive relationships were discovered between various pairs of categorical variables including the “awareness of respondents about any system/software architectural tactics or metrics that have been or are being used for evaluating architectural description models” and the opinion of respondents on the statement that “Most of the existing software architecture evaluation methods, produce qualitative results”. \( \chi^2(4) = 9.55, \) p-value = .049 < .05, among others highlighted in the full analysis document.

V. CONCLUSION AND RECOMMENDATION

Description and evaluation of software/system architecture largely depends on how it has been embraced by developers. This study reveals the great effort that is needed to increase the awareness, knowledge and use of system software architecture in industry, academic institutions, and government sectors. Various factors that encourage and hinder the embracement of architectural modelling in the software industry were also identified. The researchers recommend further research to determine how to standardize the languages used in architectural modelling to make the models compatible with other models that are already existing. There is also a need to increase awareness of system/software architecture description by documenting more literature on the subject and incorporating it in the curricula of various learning institutions.

REFERENCES

The Five Factors influencing Software Architecture Modeling and Evaluation Techniques

Hassan Almari
Australian National University,
CSIT Building 108 North road
Canberra, ACT 0200, Australia
email: hassan.almari@anu.edu.au

Clive Boughton
Australian National University,
CSIT Building 108, North road
Canberra, ACT 0200 Australia
Chairman and Director of Software Improvements
GPO Box 1928, Canberra, ACT 2601, Australia
email: clive.boughton@anu.edu.au

Abstract—Two of the most important aspects that help architects to describe, automate, and evaluate architecture artefacts with precision include the use of Software Architecture (SA) modeling languages, and the selection of SA evaluation methods. Accurate, verifiable architecture descriptions are more likely to result in successful software development outcomes. There appears to be an unnatural and significant disconnect between SA artefacts and both pre- and post-architecture development artefacts. The disconnect seems to exist for various, sometimes unrelated, reasons not all of which have yet been fully investigated. In an effort to confirm (some of) the factors that influence effective utilization of software/system architecture artefacts in the process of software/system development, the author(s) of this paper try to address the aforementioned problem by focusing on the investigation of five factors that influence SA evaluation and its automation process. These factors include: Formality of SA descriptions; modelling of SA; SA documentation; standardisation of SA; and current SA evaluation tools. Contributing to the identification of these five influential factors, and their discussion, is a section of a questionnaire which was broadly aimed at discussing matters relating to software/system architecture descriptions and evaluation in industry.

Index Terms—Automation, Documentation, Evaluation, Languages, Measurement, Software Architecture (SA), Standardization, Utilization Factors.

I. INTRODUCTION

There are many factors that influence the utilization of software/system architecture and its evaluation method in industry today. So a questionnaire was drafted to gather the experiences of software/system developers with software architecture artefacts, and identify the evaluation methods that they use in their work.

Most importantly, the questionnaire’s aim was to identify the factors that encourage or hinder the utilization of software architecture by the developers as well as their evaluation methods; and also, to point to some aspects that could improve the automation of SA. This report analyses five particular factors in an effort to arrive at applicable solutions to increase awareness and improve the utilization and automation of software/system architecture artefact in industry.

II. ANALYSIS PROCEDURE

The original questionnaire used in this research comprised 23 questions. The questions were grouped into various sections such that each section could focus on a specific objective.

The first section focused on learning more about who responded to the questionnaire in terms of their area of expertise, the sectors in which they have worked, the length of experience and their familiarity with software/system architecture. This was important in regard to analysing their responses to the main questions more accurately.

The next section pertains to the first part on which this report focuses: discovering the factors that encourage and discourage the utilization of system/software architecture by developers in their work. Finally, the third section mainly concentrated on the evaluation methods used by developers for system/software architecture artefacts. The main task here was to gather the experience of participants with evaluation methods, hence determine the main factors that could support or hinder the use of SA artefacts and their quantitative evaluation.

Both single dimensional and two dimensional analysis were used to achieve the objective outlined in this report.

Four questions out of 23 in the questionnaire were picked and analysed to draw conclusion about the factors that influence SA artefacts’ utilization as well as factors that affect automation of SA quantitative evaluation. These include Q6, Q7, Q13 and Q14. The two former ques-
tions focused on software/system architecture utilization and automation, while the latter two questions focused on software/system evaluation.

In order to draw more meaningful conclusions, an evaluation of how the various factors and opinions obtained from the responses to each individual question in the questionnaire affect, or relate to each other, was conducted. This meant carrying out a two-dimensional analysis, where selected questions were analysed as pairs in different combinations based on the study objectives. Appropriate analysis methods were used to draw conclusions on the cross-relationship between the various aspects presented by each question.

All of the four questions were multiple choice in which the researchers proposed some of the possible factors and the participants were to choose at most two of the factors according to their experience. The analysis in the single dimension thus involved tabulating the results in a frequency table to determine the percentage of respondents who chose each of the suggested (main) factors. The conclusion was arrived at by comparing all the factors according to the number of respondents who chose them. A majority percentage was translated to be a suggested main factor the choices presented. These results were also illustrated on a bar chart for ease of visualization, where the height of each bar corresponded to the count of respondents who chose the factor under consideration.

In order to determine how other factors presented in all the questions of the questionnaire affected the responses above, a two dimensional analysis was also employed as in Table I, where each matrix is a combination of two questions.

Due to the length of analysis of the four questions (Q6, Q7, Q13 and Q14) which surpasses the page limitation, only the significant results in the analysis were chosen and discussed for all the two dimensional analyses in this report. The main method used to analyse the cross-relationship in this case was cross-tabulation. Other techniques such as the Pearson \( \chi^2 \) test, independent sample t-test and ANOVA were also used to analyse the significance of the relationships.

Normality of the items were tested using Shapiro-Wilk W test as this test has the best power for a given significance, followed closely by Anderson-Darling when comparing the Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors, and Anderson-Darling test, [1]. The randomness of missing values were tested using Little’s MCAR test.

### III. RATIONALE OF THE ANALYSIS METHOD SELECTED

There were many ways that the data from the questionnaire could have been analysed in order to arrive at some conclusion. However, careful consideration was given to selecting the methods of presenting and analysing the responses to the questionnaire in order to ensure that the conclusions arrived at were accurate and reliable. The following section discusses some of the factors that were considered in choosing the methods to perform the analysis.

#### A. Individual analysis methods

For one dimensional analysis, the method chosen to analyse the responses to each question in the questionnaire depended largely on the type of question. All of the four questions analysed for this paper involved multiple choices in which the respondents were allowed to pick any two. It is important to note that there are many factors that could influence the utilization of modelling techniques in system/software architecture description. Similarly, there are many factors that could influence quantitative evaluation for software architectures. Given the diversity of the respondents who participated in the questionnaire, it could therefore be expected that the factors they identified varied widely. If the questions were left open-ended, it would be hard to analyse the responses quantitatively. For this reason, the questionnaire was designed in such a way that the most common factors were identified by the researchers based on past experience and analysis.

The identified measures discussed above enabled quantitative analysis in which the importance of a factor was gauged by the number of respondents who chose it. Thus, it was appropriate to analyse the results using a frequency table for each of the questions. A pie chart was not appropriate for representation of these results pictorially since respondents were allowed to identify more than one factor, therefore the results did not total up to 100%. For this reason, a bar chart was more appropriate.

#### B. Multi-dimensional analysis methods

Since the four questions were close-looped, one dimensional analysis alone would not have been sufficient to draw meaningful conclusions. It was necessary to identify the trend in the responses given by the various respondents based on their responses to the rest of the questions. For this purpose, all significant results from all combinations in the two dimensional analysis that included any of the four questions relevant to in this paper were analysed alongside the selected questions in the survey (see Table I), to determine the relationships that could be present, thus contributing to this study’s objectives.

For cross-comparison with questions which involved a nominal scale such as Q2–Q8, Q13 and Q14, cross tabulation was considered more appropriate. In order to determine the level of dependency between the two questions under examination in each case, a Pearson-Chi Square test was conducted on each of the pairs too.

Questions 9 and 10 as well as Q15 to Q23 involved an ordinal scale. For this category of questions, it was necessary to use independent sample t-tests to deduce a cross-relationship with the chosen question in two dimensional analysis.

In the two dimensional analysis, most of the questions did not yield any significant results regarding the relationship between them. This led to the conclusion that three
Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. security performance).

As Design Models” as the main factor that discourages the use of modelling techniques in system/software architecture description. Closely following this is “the lack of standardization between the existing architecture modelling techniques, notions and semantics”. This SA standardization challenge was identified by [2]. Furthermore, 22% acknowledged in the previous question that the use of models “makes the evaluation of stakeholders requirements...
for quality attributes possible in the early stages of the development life cycle", 26% reckon that it is “hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)”. A further 24% think that “modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money”. Finally, 16% of the respondents pointed out that “current architecture description languages (including modelling languages) are still immature”.

As can be observed in Figure 3 all the factors identified above carry some weight as hindrances to the use of modelling in software/system architecture description. All the five factors thus need to be taken into careful consideration in efforts to encourage developers to use models in their work. Although, there are a recognised effort done to improve some of the areas that hinder the use of, frameworks and models to describe and document SA, such as [3], [4], [5], [6], and [7]. However, More research and effort need to be made, to improve the utilization and usability factors of SA modeling techniques.

Comparing the responses to the two questions, we believe that the factors that discourage the use of models in SA description far outweigh the factors that may support or encourage their use. This leads us to conclude that even though the use of modelling techniques in system/software architecture description is important, there is still a lot that needs to be done to make the process more effective and easier to use for developers.

As identified above, it is clear that there is a divided view about the contribution of modelling in making the process of evaluation effective. This is because evaluation of software architecture in general, is a fundamental component of development that should be embraced by all developers. While some of the developers have embraced quantitative evaluation for SA, there are still many others who haven’t used it. This is influenced by a number of factors.

According to the analysis of responses which are shown as percentage frequency in Table II, the main factor that could support quantitative evaluation for any SA is “the availability of tools for describing and evaluating software architecture”. This is closely followed by the “use of standard language and architecture framework for describing SA”. Some attempt been done to improve the area of both factors above such as, modeling tool by [8], also the architecture description interchange language by [9], and the architectural model-based language by [10]. However, it not been spread out within industry so far.

The other factors that could support quantitative evaluation for SA include the formality level of SA description, documentation mechanism used during SA description, and the language used for describing SA.

We therefore believe that in order to influence the use of quantitative evaluation for SA, not only do the tools for describing SA need to be availed, but it is also important to pay close attention to the language that is used in describing SA in terms of its formality, standardization and nature. As seen from Table II, a total of 39 (out of 43) respondents identified factors that have to do with the language used in describing SA. Finally, it is also important to improve on the documentation of SA description.

Further, analysis on the main factors that discourage the use of quantitative evaluation for SA reveals that one of these factors is the “formality level of SA description” used. We think that some improvement and adjustment of Rushby’s four levels of formalization [11], to suit SA will help Architects to know how far they should formalize SA.

The other main factor as identified by respondents is “the language used for describing the SA”. Others include the availability of tools for describing the SA, the documentation mechanism used during SA description and the use of standard language and framework for describing SA. It is noted that two factors were identified as the major by the respondents surrounding hindrances - tools and description languages. This seems a valid expectation based on researchers preliminary investigation, showing the reliability of the results.

A closer analysis of these results with the results for the factors that could support the use of quantitative evaluation for SA shows that there is a similarity in the factors identified. As seen from Table III, the three main factors that hinder the use of quantitative evaluation in SA still revolve around the availability of tools, formality level of SA description, as well as the language used in description of SA. We therefore believe that a solution must involve

---

**TABLE II**

<table>
<thead>
<tr>
<th>Important Factors that could Support Quantitative Evaluation for Any SA</th>
<th>Responses</th>
<th>Percentage based on who responds to this question (N=43)</th>
<th>Percentage based on total responses (N=50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>7</td>
<td>16.3%</td>
<td>14.0%</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>12</td>
<td>27.9%</td>
<td>24.0%</td>
</tr>
<tr>
<td>Using standard language and architecture framework for describing SA</td>
<td>20</td>
<td>46.5%</td>
<td>40.0%</td>
</tr>
<tr>
<td>Tools availability for describing and evaluating SA</td>
<td>22</td>
<td>51.2%</td>
<td>44.0%</td>
</tr>
<tr>
<td>Documenting mechanism used during SA description</td>
<td>9</td>
<td>20.9%</td>
<td>18.0%</td>
</tr>
</tbody>
</table>
### TABLE III
**IMPORTANT FACTORS THAT COULD HINDER QUANTITATIVE EVALUATION FOR ANY SA**

<table>
<thead>
<tr>
<th>Important Factor</th>
<th>Responses</th>
<th>Percentage based on who responds to this question (N'=43)</th>
<th>Percentage based on total responses (N=50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>17</td>
<td>40.5%</td>
<td>34.0%</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>21</td>
<td>50.0%</td>
<td>42.0%</td>
</tr>
<tr>
<td>Using standard language and architecture framework for describing SA</td>
<td>5</td>
<td>11.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>Tools availability for describing and evaluating SA</td>
<td>15</td>
<td>35.7%</td>
<td>30.0%</td>
</tr>
<tr>
<td>Documenting mechanism used during SA description</td>
<td>6</td>
<td>14.3%</td>
<td>12.0%</td>
</tr>
</tbody>
</table>

### TABLE IV
**CROSS TABULATION BETWEEN RESPONDENT’S (FREQUENT USAGE OF MODELS) AND THE (MAIN FACTORS THAT DISCOURAGE THE UTILIZATION OF SA MODELLING TECHNIQUES)**

<table>
<thead>
<tr>
<th>Main Factors That DISCOURAGE the utilization of modelling techniques to describe software/system architecture</th>
<th>Never</th>
<th>Infrequently (≤10%)</th>
<th>Reasonably frequent (11% to 50%)</th>
<th>Regularly (&gt;50% and ≤80%)</th>
<th>Nearly always (&gt;80% and ≤90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Lack of standardisation between existing architecture modelling technique, notations, and semantics</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Current architecture description languages (including modelling languages) are still immature</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### TABLE V
**CROSS TABULATION BETWEEN RESPONDENT’S (YEARS OF EXPERIENCE) AND THE (MAIN FACTORS THAT DISCOURAGE THE UTILIZATION OF SA MODELLING TECHNIQUES)**

<table>
<thead>
<tr>
<th>Main Factors That DISCOURAGE the utilization of modelling techniques to describe software/system architecture</th>
<th>5–10 (years)</th>
<th>10–15 (years)</th>
<th>15–20 (years)</th>
<th>20–25 (years)</th>
<th>Over 25 (years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Lack of standardisation between existing architecture modelling technique, notations, and semantics</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Current architecture description languages (including modelling languages) are still immature</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
These associations were expected since developers who never use models to describe software architecture may not be able to point to the reasons that influence them, as opposed to those who use models more often. This extends to developers who have not stayed in the development industry for long to identify the trends in the use of models in SA description.

We therefore believe that in order to improve the process of identification of these factors, it is necessary to encourage developers to use modelling techniques in SA description more frequently. More efficiency may be obtained by choosing respondents who have gained a long-time experience in system/software development for similar surveys in the future. The results also reveal less respondents with over 25 years of experience identified the factors listed as hindrances to the use of modelling in SA description. This could mean that utilization of modelling becomes easier and more efficient as the developers stay in the industry longer, or it could mean that the percentage of respondents with over 25 years experience was too low for results to be significant.

In identifying two dimensional relationships around the factors that may discourage the utilization of modelling techniques in SA description, a null hypothesis, \( H_0: \mu_1 = \mu_2 \) was developed alongside an alternative hypothesis, \( H_a: \mu_1 < \mu_2 \) where, \( \mu_1 \) = the population mean of the selected item (the column variables) for the respondents who answered “YES” to the item “Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system” and \( \mu_2 \) = the population mean of the selected item (the column variables) for the respondents who answered “NO” to the same statement above. An independent sample t-test used to test the above hypothesis with the item “Usage of software style/pattern concepts & models during architecture development (\( \chi^2 = 30.44, \text{p-value} < .07 \)). A cross tabulation of the results is shown in Table V.

These associations were expected since developers who never use models to describe software architecture may not be able to point to the reasons that influence them, as opposed to those who use models more often. This extends to developers who have not stayed in the development industry for long to identify the trends in the use of models in SA description. This could mean that utilization of modelling becomes easier and more efficient as the developers stay in the industry longer, or it could mean that the percentage of respondents with over 25 years experience was too low for results to be significant.

In identifying two dimensional relationships around the factors that may discourage the utilization of modelling techniques in SA description, a null hypothesis, \( H_0: \mu_1 = \mu_2 \) was developed alongside an alternative hypothesis, \( H_a: \mu_1 < \mu_2 \) where, \( \mu_1 \) = the population mean of the selected item (the column variables) for the respondents who answered “YES” to the item “Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system” and \( \mu_2 \) = the population mean of the selected item (the column variables) for the respondents who answered “NO” to the same statement above. An independent sample t-test used to test the above hypothesis with the item “Usage of software style/pattern concepts & models during architecture development (\( \chi^2 = 30.44, \text{p-value} < .07 \)). A cross tabulation of the results is shown in Table V.

These associations were expected since developers who never use models to describe software architecture may not be able to point to the reasons that influence them, as opposed to those who use models more often. This extends to developers who have not stayed in the development industry for long to identify the trends in the use of models in SA description. This could mean that utilization of modelling becomes easier and more efficient as the developers stay in the industry longer, or it could mean that the percentage of respondents with over 25 years experience was too low for results to be significant.

In identifying two dimensional relationships around the factors that may discourage the utilization of modelling techniques in SA description, a null hypothesis, \( H_0: \mu_1 = \mu_2 \) was developed alongside an alternative hypothesis, \( H_a: \mu_1 < \mu_2 \) where, \( \mu_1 \) = the population mean of the selected item (the column variables) for the respondents who answered “YES” to the item “Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system” and \( \mu_2 \) = the population mean of the selected item (the column variables) for the respondents who answered “NO” to the same statement above. An independent sample t-test used to test the above hypothesis with the item “Usage of software style/pattern concepts & models during architecture development (\( \chi^2 = 30.44, \text{p-value} < .07 \)). A cross tabulation of the results is shown in Table V.
opment, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture evaluation” showed a significance at the 5% level, $t = -1.8$, p-value < .05, power = .60. To achieve 80 percent power a sample of $n_1 = n_2 = 51$ needed. This shows a lower population mean for those who answered “Yes” to the hypothetical statement compared to those who answered “No”.

This also shows the contribution of (software patterns) which were evaluated in the previous questionnaire in an effort to minimize the hindrances to the utilization of modelling techniques in system/software architecture description. Also, the result above could be due to the respondents’ disagreement to the second part of the pattern statement, which is (Usage of software style/pattern concepts & models during architecture development, increases the utilization of modelling description languages, BUT decreases the simplicity of the architecture evaluation).

A significant relationship was also noticed between the most important factors that could SUPPORT quantitative evaluation for any software architecture (SA) and the opinion that “Usage of software style/pattern concepts & models during architecture development, increases the utilization of modelling description languages, BUT decreases the simplicity of the architecture evaluation”. $\chi^2(20) = 34.36$, p-value < .05. This further reveals the contribution of software style/patterns on both SA evaluation as well as the use of modelling techniques in SA descriptions.

### TABLE VII
**Pearson Chi-square Test Results for Analyses of Questions Q13 and Q22.**

<table>
<thead>
<tr>
<th>Chi-square</th>
<th>df</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.75</td>
<td>20</td>
<td>.059</td>
</tr>
</tbody>
</table>

Similarly, the evaluation of the statement that “there is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers” around the null hypothesis, $H_0$: $\mu_1 = \mu_2$ and alternative hypothesis, $H_a$: $\mu_1 \neq \mu_2$ where $\mu_1$ is the population mean of the selected item (the column variables) for the respondents who answered “Yes” to the item “Hard to evaluate architecture models against any quality attributes (e.g. Security, performance)” and $\mu_2$ is the population mean of the selected item (the column variables) for the respondents who answered “No” to the item “Hard to evaluate architecture models against any quality attributes (e.g. Security, performance)” showed a significantly lower population mean for those who responded “Yes” to the item than those who responded “No”, $t = -1.96$, p-value = .028 < .05, power = .46. To achieve 80 percent power, a sample size of $n_1 = n_2 = 76$ needed. The results are arrived at by considering a left-tailed population.

Also, more analysis reveals that the relationship was also detected for the factor “Most of the existing software architecture evaluation methods, produce qualitative results”. $t = 1.94$, p-value = .03 < .05, power = .72. To achieve 80 percent power, a sample size of $n_1 = n_2 = 34$ needed.

In a similar way, a significant relationship was established at 6% level, between the most important factors that could SUPPORT quantitative evaluation for any software architecture (SA) and the opinion about the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” $\chi^2(20) = 30.75$, p-value < .06. The results of the cross-comparison are shown in Table VI, VII. This relationship is an indication of the effect of the use of current technology on factors that could support quantitative evaluation of SA, as well as the use of modelling in SA descriptions.

Furthermore, independent sample t-tests performed to determine the association in the agreement to the statements “Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DODAF, RUP/4+1) increases the reliability, standardization, and reusability of the resulting architecture” and “Usage of software style/pattern concepts & models during architecture development, increases the utilization of modelling description languages, BUT decreases the simplicity of the architecture valuation” as compared with the responses to the main factors that encourage the utilization of modelling techniques to describe software/system architecture. Both statements’ relationships (above) with all encouraging factors are insignificant at the level of 5%. This independence in opinion is expected and it shows the genuineness of the responses given in this study to evaluate the factors that influence the use of modelling as well as quantitative evaluation in SA description.

In order to determine the cross-relationship between the responses with ordinal items in Q15 to 23, null and alternative hypotheses were formed equating the pairs of items. Interesting relationships were noted. For example, there was a significant relationship at the 5% level ($t = 2.2$, p-value < .05), power = .74, pointing out that the population mean of the item “Reliable tools are important for developing/or evaluating software/system architectures” is significantly higher for the respondents who replied “Yes” to the item “It makes the designers/programmers’ job much easier”. This is an expected relationship, showing the zeal of developers to find effective tools to make their work easier, thus supporting the use of models in SA description.

Similar relationships were noted between the population means for the item “Architecture is design, but NOT all
“design is Architecture” which is significantly higher for the respondents who replied “Yes” to the item “Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development lifecycle”.

Generally, the responses to some of the items in the ordinal questions have identified critical factors that influence the use of models and quantitative evaluation in SA description, which required more attention from software engineers to improve the current state of the art in the domain of SA. As a result, the above improvements will impulse SA automation process in the industry.

V. CONCLUSION AND RECOMMENDATION

This study has helped to uncover and then confirm some of the most important factors that encourage or hinder the utilization of models in SA description as well as the use of SA quantitative evaluation. These factors include the use of SA standardized frameworks and description languages, the level of SA description formality, interface mechanism (integration) between SA artefacts and other phases within the development lifecycle (e.g requirements and design). Also, tools for describing SA are important, as well as SA documentation process.

We believe that the above factors are key to improve SA deliverables, evaluation processes, and automation procedures. It is of paramount importance to encourage the use of models in the description of system architecture due to the fact that have been identified above. Unfortunately, most of the developers don’t often use models in SA descriptions, due to the reasons that have been identified. We recommend further research to focus on the solutions that could be implemented for the hindrances, while devising ways of encouraging the use of models by enhancing the encouraging factors identified above.

As identified by the two dimensional analysis, most of the experiences of developers with the use of models in their work are influenced by other factors. For example, it is clear that people have a need for tools which can make their work easier in software development. It has been clearly identified in this research that one of the most important factors that encourages the use of models in software architecture description is the ability to ease the developers’ work. Thus, more emphasis needs to be given to the use of models in the developers’ work. The first step to achieving this and encouraging more developers to embrace it, is to find solutions to the hindrances identified above. This is also the case for the use of quantitative evaluation of software architectures.

REFERENCES


Appendix

B

Complementary background information for Chapter 2

B.1 Introduction

This appendix contains some sections, which include information, figures, and tables that might support my arguments, analysis, and discussion presented in the background chapter, (Chapter 2). The following sections of this appendix does not designed as an isolated entities that can be understandable without their links to the main related sections on Chapter 2 where they have been cited.

B.2 MDA Advantages and Disadvantages

![Diagram](image)

Figure B.1: MDA advantages, *After* Flint [2008].
B.3 More about ADLs

The architecture of a software application is a high level description of its structure made up of individual independent components and is an important part of the success of the system. Along with the system structure, the architecture also describes features and properties such as interactions between the components, security and auditing frameworks, performance variables, accuracy, failure resistance, compliance to standards and quality. Several factors, like reusability, flexibility of modification and requirement compliance, need to be taken into account while choosing the architecture of a system to ensure that the system is reliable and scalable. Inappropriate design of the architecture is responsible for several software development related issues and sometimes can even lead to the failure of the entire process.

Despite being extremely critical for the success of the system, most of the architecture descriptions are based on informal methods without any tool support and hence are often vague, inconsistent, unverifiable and open to misinterpretation. Many of the boundary assumptions made during the architecture design are overruled during the development process. To overcome all these problems, formal methods and notations have been proposed some common examples of which include ADLs and DSLs.

1. Components: These are the individual computational objects or elements of the system and are the main participating entities of the system. A system is made up of several simple or composite components interacting with each other.
2. Operators: The operators are used to describe the relationships between the individual components and to represent the connectivity between components.
3. Patterns: The patterns describe the way in which the components are connected to each other. These are more commonly known as design patterns and are templates for the design. Sometimes, the patterns are also known by other names, like behavioural patterns and architecture patterns, but overall the concept remains the same. These patterns are reusable and each pattern aims to solve a specific design problem. The actual instances of the design are created during the design phase. The template provides the blueprint for the element properties and interactions Qin et al. [2008]. The patterns can also be called as concepts instead of templates, which can evolve and produce new patterns designed specifically to solve certain problems. This element emphasize the robust relationship between Software Pattern (SP) and Software Architecture (SA).
4. Closure: It symbolises the data encapsulation property and is an entity, which has hidden data and properties within it. The entity can be used as a normal function to achieve certain goals.
5. Specification: This includes the functional rules apart from other type of specifications like performance and fault-tolerance.
Apart from the above mentioned elements, there are also other views related to ADLs along with their elements, like properties, ports, roles, red-maps and bindings, which have been proposed by other researchers like Garlan, Monroe and Wile. Even though the different ADLs available have different capacities and functionalities, they are tied together on the basis of a common design concept of architectural description. According to Shaw et al. [1996], Medvidovic et al. [1997], the main elements of this basis are as follows:

1. **Components** are the individual elements of the system and connectors represent the way they interact with each other.
2. **A system** is made up of components and connectors. A system may be made up of several hierarchical sub-systems each of which is made up of components and connectors. A component can also have its own architectural description known as sub-architecture used to describe the finer details of the component if needed. The overall structure of the system can be described independent of the components and connectors that make it up.
3. Information about the system characteristics is stored in fields known as **properties**. Each ADL has its own set of important properties but all of them provide some way of storing and analysing some extra-customised properties as well. For example, there are properties that hold calculated system throughput based on performance measures, Shaw et al. [1996].
4. **Constraints** are the system boundaries, which remain constant over time and system evolution. Some typical constraints are the range of allowable values and topology. For example, some administrative applications might restrict the number of clients that can connect to the server to a minimum value to ensure changes made by one administrator are not being overwritten by another.
5. **Architectural styles** basically define the way elements are composed and connected to each other. For example, data-flow architecture is composed of pipes and lines connected together with specific graphical notations. Similarly, blackboard architecture is comprised of knowledge banks (resources) and systems working together with a central control mechanism. A few architectural styles even provide a framework library that can be customised according to the needs of an application. Some typical examples are the resource management style schema, networking style schema.

### B.3.0.1 ADL elements

The usage of ADL is similar to the use of any other programming language wherein the programming can be done either by entering data in the form of text or by using supporting graphical tools. According to the classic theory of Shaw et al. [1996] proposed in Shaw et al. [1996], the main elements of an ADL are defined as components, operators, patterns, closure and specification; each of which is outlined below:

### B.3.0.2 Design goals of ADLs

As mentioned before, there are several ADLs available in the market. However, before choosing a particular ADL, it is important that the ADL meets certain design goals and criteria. First and foremost, the ADL should support dynamic analysis of the architectural structure during run time. All the elements of the system, like components, connectors and configuration elements, should be modifiable dynamically. The architectural structure may be made up of a combination of components, connectors and configuration elements, none of which is mandatory. In any architecture, the components and connectors interact with the external environment through interfaces. The connector elements also interact with the components and connect and disconnect with them regularly through these interfaces. The configuration files hold design and runtime information about the structure and hierarchy of the architecture and are also used for validation purposes.

Secondly, an ADL should allow hierarchical system descriptions so that the system can be described at multiple levels in terms of several interrelated subsystems with separate configuration settings for each subsystem. This makes the overall system easier to understand and flexible. The ADL should also allow styles to be defined and extended as needed. This allows the system to be described at as abstract a level as needed.

Thirdly, an ADL should use formal methods and tools for system verification purposes irrespective
APPENDIX B. COMPLEMENTARY BACKGROUND INFORMATION FOR...

Table B.1: ADL design goals

<table>
<thead>
<tr>
<th>ADL Language</th>
<th>Design goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2 Taylor et al. [1996]</td>
<td>Primarily used in Graphical User Interface (GUI) development and supports reusability in development. GUI development forms a large fraction of any software development cycle, yet reusability is a rare functionality in GUI design. C2 also allows dynamic component changes.</td>
</tr>
<tr>
<td>WRIGHT Allen et al. [1997]</td>
<td>This provides support for customised vocabularies along with regular functionalities like specification, analysis, description, validation etc. It provides special attention towards the specification of interactions between the components of the system and allows for the analysis of these interactions at a high level.</td>
</tr>
<tr>
<td>ACME Garlan et al. [1997]</td>
<td>This is primarily an interchange ADL. See (Section 2.3.4 - Chapter 2)</td>
</tr>
<tr>
<td>Darwin</td>
<td>This is primarily used to describe systems which are made up of varying components and interactions in a declarative manner and supports distributed messaging. The notations for these diverse systems are kept as general as possible.</td>
</tr>
<tr>
<td>π-ADL Oquendo [2004b]</td>
<td>This is meant primarily for dynamic and mobile architectures wherein the dynamic architecture components can change in real-time.</td>
</tr>
<tr>
<td>xADL Dashofy et al. [2005]</td>
<td>This enables faster ADL constructions and provides better scalability.</td>
</tr>
<tr>
<td>KDL He et al. [2005]</td>
<td>This is a domain specific ADL used mainly for E-commerce (EC) knowledge description.</td>
</tr>
<tr>
<td>Aesop</td>
<td>This ADL provides style support.</td>
</tr>
<tr>
<td>Meta-H</td>
<td>This is a domain specific ADL used in the description of real time avionics control software</td>
</tr>
<tr>
<td>Rapide</td>
<td>This provides design simulation and analysis tools for architecture descriptions. The simulation outputs can also be analysed using the tools.</td>
</tr>
<tr>
<td>SADL</td>
<td>This lays the foundation for refining architectural descriptions.</td>
</tr>
<tr>
<td>UniCon</td>
<td>This provides support for combined heterogeneous component and connectors and uses a high level compilation tool.</td>
</tr>
</tbody>
</table>

of whether it is a general purpose ADL or domain specific. General purpose ADLs should ensure that the design and the features are kept as simple and minimal as possible. If the ADL is domain specific, it should also allow domain related notations and terms to be employed within it to support specific functionalities.

Overall, the purpose of ADL is to provide an extensible framework as well as syntax library for describing various types of system architectures along with supporting tools for various functions like parsing, display, compilation, simulation and any other architecture description related activities. Table B.1 describes some common ADLs, their design goals and their behaviour. There are also other parameters for ADL comparison like behaviours and supported interfaces. Some of these common characteristic comparisons are illustrated by Table B.2. Both tables mentioned above are a critique summary about the current state of the art, such as Bass et al. [2013], Qin et al. [2008], Garlan et al. [2006b], Dissaux et al. [2005], Oquendo [2004b], Bosch et al. [2002], Medvidovic et al. [2000], Medvidovic et al. [1997], and Garlan et al. [1997].

Table B.2: ADL characteristics

| Modelling Connectors | Some of the common ADL model connectors are ACME, C2, WRIGHT, xADL and π-ADL model connectors. Connectors, which are represented by Darwin, where the bindings are fixed and cannot be sub typed or reused. They use inline modelling and are generally unnamed. |
**Interfaces**

Connector interfaces are generally available only for connectors modelled at the first level of hierarchy. There is no difference in component and connector interfaces except in the way they are referred. In some ADLs like ACME and WRIGHT connector interfaces are called roles, in some others like xADL they are called interfaces. All of them allow type definitions for these interfaces. In WRIGHT ADL, the interaction protocols at each stage in the Communicating Sequential Processes (CSP) remains uniform.

**Types**

Connector types and instances are considered to be different only in those ADLs in which the connectors are modelled at the first level of hierarchy and hence Darwin does not come into picture whereas ACME, C2, WRIGHT, xADL, \(\pi\)-ADL etc. distinguish between the two. These ADLs model connector types on the base of interaction protocols. Also, ACME and xADL allow connector instances, which do not have a type definition.

‘ACME and \(\pi\)-ADL’ provide parameter support which means that the connector signatures can be specified in a customised format along with any applicable constraints. This allows for flexible interfaces. Similarly, WRIGHT allows connector parameterization based on its behaviour.

**Behavioral Specifications**

C2 allows certain specifications and semantics through the use of invariants. WRIGHT also supports component and connector behavioural specifications through a language called CSP. Darwin and \(\pi\)-ADL use \(\pi\)-calculus to describe specifications and properties. \(\pi\)-calculus is an important concept in the description of behaviour specifications. Random complex specifications are treated as continuous annotations in ACME and xADL. The behaviour specifications are different for each type of ADL.

**Modelling, Configurations, Compositionality**

Most ADLs allow hierarchical descriptions of systems with subsystem configurations. The hierarchies can be described in the same manner as configurations. In case of Darwin such systems are modelled as composite components instead of using configuration elements. ACME supports hierarchical description for both components and connectors and calls it representations. WRIGHT also allows composite components and connectors along with configuration support. A composite component is again represented by a separate configuration segment.

In \(\pi\)-ADL, components and connectors can be used to construct further composite elements, components or connectors and vice versa. Also, it supports decomposition and recomposition.

**Implementation**

Only some ADLs provide implementation support and even those are tightly coupled to some programming language. For example, Darwin allows deployment of components developed in C++. C2 supports elements developed using C++, Ada, and Java, while xADL supports elements coded using Java. This dependency on language sometimes reduces their flexibility and makes the usage limited to certain platforms.
Table B.2: (continued)

| Dynamism | Darwin, C2 and \(\pi\)-ADL all allow the system and the configuration elements to change at runtime. Darwin supports this by virtue of lazy and dynamic instantiation where instantiation is done as and when needed. This is similar to the lazy loading concept in Object Oriented analysis. C2 has its own architecture modification language which allows dynamic modification of elements of the architecture at runtime. The modification language known as AML supports insertion, deletion and rewiring of new elements at specific locations with the help of 4 pre-defined functions called AddComponent, RemoveComponent, Weld and Unweld. \(\pi\)-ADL allows dynamic behaviour based on certain runtime parameters and conditions Oquendo [2004a] and also supports description of mobile architecture Qin et al. [2008]. However, since most of the information is known only at design time and not actually at runtime, most ADLs are static which diminishes their applicability and efforts are on to create new and more dynamic ADLs. For example, the new versions of xADL might include new modelling constructs which will support distributed and dynamic architectures. |

B.3.0.3 The Most ADLs that are still supported

Table B.3: The Most Known ADLs that still supported: the table contains also approaches which are considered non-conventional ADLs, since they might neglecting fundamental aspects, After Rech et al. 2009, pp 267.

<table>
<thead>
<tr>
<th>ADL</th>
<th>Born Data</th>
<th>Tools</th>
<th>Still Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rapide</td>
<td>1990</td>
<td>Rapide</td>
<td>NO</td>
<td>ADL and simulation</td>
</tr>
<tr>
<td>Darwin</td>
<td>1991</td>
<td>LTSA + SAA</td>
<td>YES</td>
<td>Focus on dynamic SA</td>
</tr>
<tr>
<td>Weaves</td>
<td>1991</td>
<td>Weaves</td>
<td>NO</td>
<td>Data-flow-architectures with high-volume of data</td>
</tr>
<tr>
<td>Adage</td>
<td>1992</td>
<td>–</td>
<td>NO</td>
<td>Avionics navigation and guidance Architecture Description</td>
</tr>
<tr>
<td>LILEANNA</td>
<td>1993</td>
<td>LILEANNA</td>
<td>NO</td>
<td>Modeles connection language</td>
</tr>
<tr>
<td>MetaH &amp; MetaS</td>
<td>1993</td>
<td>MetaH</td>
<td>YES</td>
<td>ADL for avionic domain</td>
</tr>
<tr>
<td>ArTek</td>
<td>1994</td>
<td>–</td>
<td>NO</td>
<td>Non conventional ADL</td>
</tr>
<tr>
<td>Resolve</td>
<td>1994</td>
<td>Resolve</td>
<td>NO</td>
<td>Focus on Components Specification</td>
</tr>
<tr>
<td>Wright</td>
<td>1994</td>
<td>Wright</td>
<td>NO</td>
<td>Focus on communications</td>
</tr>
<tr>
<td>ACME</td>
<td>1995</td>
<td>ACMESstudio Armani</td>
<td>YES</td>
<td>Interchange Language between ADLs</td>
</tr>
<tr>
<td>SADL</td>
<td>1995</td>
<td>Sadl tool</td>
<td>NO</td>
<td>Focus on Refinement</td>
</tr>
<tr>
<td>UniCon</td>
<td>1995</td>
<td>Unicon</td>
<td>NO</td>
<td>Focus on connectors and Styles</td>
</tr>
<tr>
<td>C2 SADEL &amp; C2 AML</td>
<td>1996</td>
<td>Dradel, SAAGE Arch-Studio</td>
<td>NO</td>
<td>ADL based on C2 style</td>
</tr>
<tr>
<td>GenVoca</td>
<td>1996</td>
<td>P3</td>
<td>NO</td>
<td>ADL based on C2 style</td>
</tr>
<tr>
<td>Fujaba</td>
<td>1997</td>
<td>Fujaba</td>
<td>YES</td>
<td>Non conventional ADL</td>
</tr>
<tr>
<td>Jacal</td>
<td>1997</td>
<td>Jacal 2</td>
<td>YES</td>
<td>Focus on prototyping SA</td>
</tr>
<tr>
<td>Koala</td>
<td>1997</td>
<td>Koala tools</td>
<td>YES</td>
<td>ADL for product families</td>
</tr>
<tr>
<td>Little-JIL</td>
<td>1998</td>
<td>Little-JIL 1.0</td>
<td>NO</td>
<td>Non conventional ADL</td>
</tr>
<tr>
<td>Maude</td>
<td>1998</td>
<td>Maude 2.0</td>
<td>YES</td>
<td>Non conventional ADL</td>
</tr>
<tr>
<td>ADML</td>
<td>2000</td>
<td>ADML Enabled Tools</td>
<td>YES</td>
<td>XML-based ADL</td>
</tr>
<tr>
<td>xArch/xADL</td>
<td>2000</td>
<td>xADL 2.0</td>
<td>YES</td>
<td>XML-based ADL</td>
</tr>
<tr>
<td>AADL</td>
<td>2001</td>
<td>Osate</td>
<td>YES</td>
<td>Embedded real-time systems / Avionics systems</td>
</tr>
<tr>
<td>xArch/xACME</td>
<td>2001</td>
<td>ACMEStudio</td>
<td>YES</td>
<td>ACME in XML</td>
</tr>
</tbody>
</table>
### B.3. MORE ABOUT ADLS

<table>
<thead>
<tr>
<th>ADL</th>
<th>Year</th>
<th>Tool/Prototype</th>
<th>Supported</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC/ADL</td>
<td>2002</td>
<td>ABC tool (prototype)</td>
<td>YES</td>
<td>ADL for component composition</td>
</tr>
<tr>
<td>Prisma</td>
<td>2002</td>
<td>PrismaCase</td>
<td>YES</td>
<td>Component-based systems</td>
</tr>
<tr>
<td>DAOP-ADL</td>
<td>2003</td>
<td>DAOP-ADTools</td>
<td>YES</td>
<td>Component and Aspect-based ADL</td>
</tr>
</tbody>
</table>
Appendix

Database application

As extra information for the readers, this appendix includes the Database (DB) technical description, extra snapshots, development changes, and difficulties, which seems to appropriate to be here rather than in the main body of the thesis.

C.1 Technical description

- The application has been developed using Microsoft .NET Framework 3.5 and hosted on high performance Internet Information Service (IIS).
- The programming language is C#.
- Implementation of ITIL framework.
- For storing data SQL server 2005 is used as DB.
- Basic HTML is used for presentation and as user interface.
- Real time display of data.
- All the data processing is done in SQL Server 2005.
- Hosted on a state-of-the-art infrastructure.
- From front end store procedure SP or simple query is used to fetch or update the data in SQL Server.
- For making connection to database, i.e. SQL, first of all once connection element has to created syntax:

  ```csharp
  SqlConnection con = new SqlConnection(@"Server = TOSHIBA-PC\MSSMLBIZ;
  Database=master; Integrated Security = SSPI")
  ```

- After connection object is created command object is used to instruct SQL whether SPor general query is being processed.
- Once connection is made query or SP is processed by SQL server, i.e., DB and result is returned. After processing the result the connection is closed.

C.1.1 DB tables

1. Pattern table – MstPattern

<table>
<thead>
<tr>
<th>txtPatName</th>
<th>txtOtherName</th>
<th>txtContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR(SIZE)</td>
<td>150</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- txtPatName will be primary key.
- txtPatName will be used as foreign key.

2. Quality table – MstQuality
APPENDIX C. DATABASE APPLICATION

<table>
<thead>
<tr>
<th>txtQAName</th>
<th>txtQADefinition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varchar(150)</td>
<td>Varchar(1000)</td>
</tr>
</tbody>
</table>

- `txtQAName` will be primary key.
- `txtQAName` will be used as foreign key.

3. Relation definition table – MstPatQualRelation

<table>
<thead>
<tr>
<th>txtRelType (varchar(50))</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
</tbody>
</table>

- `txtRelType` will be primary key.
- `txtRelType` will be used as foreign key.
- Find out size of `txtRelType` individual text column.
- Check whether relation type can in future.

4. Matrix List – MstMatrix

<table>
<thead>
<tr>
<th>txtMatrixName</th>
</tr>
</thead>
<tbody>
<tr>
<td>varchar(500)</td>
</tr>
</tbody>
</table>

- `txtMatrixName` will be primary key.
- `txtMatrixName` will be used as foreign key.

5. Quality Pattern relation table – TrnPatQualRel

<table>
<thead>
<tr>
<th>txtMatrixName</th>
<th>txtPatName</th>
<th>txtQAName</th>
<th>txtRelType</th>
<th>txtComment</th>
</tr>
</thead>
<tbody>
<tr>
<td>varchar(500)</td>
<td>varchar(150)</td>
<td>varchar(150)</td>
<td>varchar(50)</td>
<td>varchar(5000)</td>
</tr>
</tbody>
</table>

- `txtMatrixName` will be used as foreign key.
- `txtPatName` will be foreign key.
- `txtQAName` will be foreign key.
- `txtRelType` will be foreign key.

6. Pattern categorization table:

- `spAddMatrix` – SP to add matrix details.
- `spAddPattern` – SP to add pattern details.
- `spAddQuality` – SP to add quality details.
- `spCheckPattern` – SP to check whether pattern details exist or not.
- `spCheckQuality` – SP to check whether quality details exist or not.
- `spCheckRelation` – SP to check relation between pattern and quality.
- `spGetConflict` – SP to fetch all conflicting relationship.
- `spGetMatrixDetails` – SP to get matrix details.
- `spGetPattern` – SP to get pattern details.
- `spGetPatternCommentSearch` – SP to fetch comments for pattern.
- `spGetPatternMat` – SP to get pattern and matrix details for specific matrix id. passed.
- `spGetPatternQualitySearch` – SP to fetch comments for quality.
- `spGetPatternSearch` – SP to get details for pattern id. passed.
- `spGetQuality` – SP to get quality details.
### C.1. TECHNICAL DESCRIPTION

- spGetQualityDefSearch – SP to fetch comments for quality.
- spGetQualityPatterSearch – SP to fetch comments for quality.
- spGetQualitySearch – SP to search quality.
- spPatternQualMatRelation – SP to get quality and matrix details.
- spUpdateRelation – SP to update relation.

### C.1.2 Database Snapshots

This section introduces some of the database pages. Each Figure with a title that is sufficient to convey the snapshot meaning.

![Create New Relation](image)

Figure C.1: Create new relation between SP and QA, Pattern information page.
APPENDIX C. DATABASE APPLICATION

C.1.3 Brief description of development changes and difficulties

During the database development process, a number of changes to the requirements of the application have occurred, as well as a number of difficulties have been notable. These are briefly listed as below:

Changes

- Create drop-down menu to navigate all the database pages easily
- Solid grid-lines for the matrices.

• Merge three functions (with three steps), which are adding pattern, adding quality attribute, and adding matrix, in one page titled ‘Create New Relation’
• Edit Page: Put the frame and add title on the frame.
• Increase the width for quality definition.
• Increase the width for pattern definition.

**Difficulties**

• Creation of individual matrix display.
• Pushing data from excel sheets into DB tables.
• Fixing general matrix problem and duplications.
• Creating pattern and quality table displays.
• Enabling mouse over property for patterns and qualities names.

Table C.1: Database application revisions.

<table>
<thead>
<tr>
<th>Ver. Rev.</th>
<th>Date</th>
<th>Author</th>
<th>Reviewers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0a</td>
<td>3-Apr-2011</td>
<td>Hassan Almari</td>
<td>Hassan</td>
<td>Draft Version</td>
</tr>
<tr>
<td>0.0b</td>
<td>9-May-2012</td>
<td>Hassan Almari</td>
<td>Hassan</td>
<td>Draft Version</td>
</tr>
<tr>
<td>1.0</td>
<td>1-Dec-2014</td>
<td>Hassan Almari</td>
<td>Hassan</td>
<td>Draft Version</td>
</tr>
<tr>
<td>1.1</td>
<td>24-Oct-2017</td>
<td>Hassan Almari</td>
<td>Hassan</td>
<td>Final (with some bug fixes)</td>
</tr>
</tbody>
</table>
D

Complement information for the SPs Survey

D.1 Introduction

This appendix represents supportive materials to Chapter 4, where tables, figures, and sections are ordered based on their citation on the main chapter, in order to facilitate their traceability.

D.2 Rationale of the Two-dimensional analysis method used – by details

The following table explains and justify reasons behind our selection of the analysis methods that were used for Two-dimensional analysis.

Table D.1: Two dimensional matrices analysis methods.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Analysis methods used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 and Q5</td>
<td>Student’s t-test was used to see whether developers with expertise in specific fields used software styles/patterns more frequently than who didn’t have same expertise. Choice of t-test either equal or unequal variances are fixed by a Levene’s F-test of equality of two variances. If P-value ≤ 0.05, the test is significant (5% level of significance). A bar chart represents the mean of the two groups.</td>
</tr>
<tr>
<td>Q1 and each of Q17 to Q20</td>
<td>A one sample t-test was used, to test whether the mean value for the statements was ≥ 3 (neutral value), to determine the general agreement of the developers who have the expertise in specific software development field.</td>
</tr>
<tr>
<td>Q2 and each of Q5, Q8; Q3 and each of Q4, Q5; Q6 and Q16</td>
<td>A cross tabulation with Chi-square test was used for testing independence of two categorical variables. Stacked bar charts were created to visualise the relationship between both criteria more easily.</td>
</tr>
</tbody>
</table>
### D. Complement Information for the SPS Survey

<table>
<thead>
<tr>
<th>Question(s)</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2 and each of Q17 to Q20</td>
<td>A one sample t-test was used, to test whether the mean value for the statements was $\geq 3$ (neutral value), to determine the general agreement of developers who gained their experience in different software development sectors. A stacked bar chart was created, with each bar representing each sector and the mean stacked in each bar.</td>
</tr>
<tr>
<td>Q3 and each Q14 to Q16; Q5 and each of Q6, Q7, Q16; Q3 and each Q14 to Q16; Q5 and each of Q6, Q7, Q16</td>
<td>A cross tabulation with Chi-square test was used for testing independence between two categorical variables and to determine if this dependence is statistically significant.</td>
</tr>
<tr>
<td>Q3 and each of Q17 – Q20</td>
<td>Pairs of ordinal variables were arranged in a scatter plot and Pearson’s correlation coefficients calculated to determine the correlation between variables. T-tests were also used to test the significance of correlation coefficients. P-values $\leq 0.05$ signify a significant relationship between both question variables.</td>
</tr>
<tr>
<td>Q4 and Q13</td>
<td>A relationship table between both questions was created and percentages/proportion used to describe the relationship between them.</td>
</tr>
<tr>
<td>Q4 and each of Q17 – Q20</td>
<td>Descriptive statistics with bar chart were used, with bars representing the mean value of each statement for developers who were aware of software styles/patterns. Error bars (95% confidence interval) were also displayed for each of the statements, to determine whether there is a general equality in means for the statements.</td>
</tr>
<tr>
<td>Q5 and Q8; Q5 and Q9; Q5 and Q14; Q5 and Q15</td>
<td>Independent sample t-tests were used, for testing equality of two population means corresponding to two groups.</td>
</tr>
<tr>
<td>Q5 and each of Q17 – Q20</td>
<td>The two options (Those who “used software styles/patterns infrequently or not (less than 15%)” and those who “used software styles/patterns reasonably frequently to always (more than 15%)”) were analysed using t-tests to determine the equality of two population means.</td>
</tr>
<tr>
<td>Q6 and Q16; Q7 and Q16</td>
<td>A cross tabulation with Chi-square test was used for testing the independence of the two categorical variables.</td>
</tr>
<tr>
<td>Q6 and each of Q17 – Q20</td>
<td>T-tests were used to see whether the population mean for each of the statements was equal to a hypothesised value of 3 (neutral value) for each possible response (“the main factors that discourage developers from the utilizing software patterns”).</td>
</tr>
<tr>
<td>Q7 and each of Q17 – Q20</td>
<td>T-tests were used to see whether the population mean for each of the statements was equal to a hypothesised value of 3 (neutral value) for each possible response (“the main factors that encourage developers from the utilizing software patterns”).</td>
</tr>
</tbody>
</table>

### D.3 Two dimensions supportive analysis

This section includes all information, tables, and figures that should support the 2-dimension descriptions demonstrated in Chapter 4.
D.3. TWO DIMENSIONS SUPPORTIVE ANALYSIS

D.3.1 Analyses of (Q1 and each of Q17–Q20)

The highest mean out of all the statements was for “Requirements elicitation/modelling/analysis”, the lowest being for the “Documentation” field of expertise. Means for all statements (except “Documentation”) were significantly higher than 3 at (1% level of significance). The developers with expertise in “Documentation” either disagreed or were neutral about the statements. For example, the statement “Identifying standard quality attribute definitions within current pattern references is critical for comparing the same patterns against the quality attribute they possess”, they disagreed with it at (5% level of significance), however they agreed with the other three statements at (5% level of significance).

In general, the results mostly indicated a common agreement about the four statements, as shown in Table D.2.

D.3.2 Analyses of (Q2 and both (Q5 and Q8)

Table D.3: Cross tabulation of three work sectors with software styles/patterns usage and consideration of quality attributes and Chi-square test of independence between attributes.

<table>
<thead>
<tr>
<th>How often do you use software styles / patterns during your work?</th>
<th>In which of the following sectors have you gained most of your general software development experience</th>
<th>Chi-square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Academy</td>
<td>Industry</td>
</tr>
<tr>
<td>Never</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Infrequently (&lt;10%)</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Reasonably frequently (&gt;15% and &lt;50%)</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Regularly (&gt;50% and &lt;80%)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Nearly always (&gt;90%)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>During your selection of patterns did you care about or consider quality attributes?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No - Why?</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Yes - Why?</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

D.3.3 Analyses of (Q2 and each question from Q17 to Q20)

The highest mean for each of the first three statements comes from those with expertise coming from “academia”, followed by “industry” then “government”.

317
D.5. COMPLEMENT INFORMATION FOR THE SPS SURVEY

Table D.2: Descriptive statistics and comparison of the population mean to a hypothesized neutral value of 3, using t-tests for each of the statement.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Documentation</th>
<th>Architecture</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Project Management</th>
<th>Requirements</th>
<th>Project Management</th>
<th>Requirements</th>
<th>Project Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>3.9</td>
<td>3.8</td>
<td>4.1</td>
<td>4.0</td>
<td>4.1</td>
<td>4.1</td>
<td>4.1</td>
<td>4.0</td>
<td>3.8</td>
<td>4.1</td>
</tr>
<tr>
<td>SD</td>
<td>.6</td>
<td>.9</td>
<td>.7</td>
<td>1.0</td>
<td>1.1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.2</td>
<td>1.0</td>
</tr>
<tr>
<td>p-value</td>
<td>.001</td>
<td>.141</td>
<td>.001</td>
<td>.002</td>
<td>.003</td>
<td>.000</td>
<td>.001</td>
<td>.001</td>
<td>.044</td>
<td>.022</td>
</tr>
</tbody>
</table>

Table Key:
- Mean
- Standard Deviation
- p-value corresponding to a t-test with null hypothesis \( \mu \leq 3 \) and alternative hypothesis \( \mu > 3 \).

Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field.

Identifying standard quality attributes defined within the current pattern references is critical for comparing the same patterns against the quality attributes they possess.

Studying the relationships between patterns and quality attributes based on the current reliable software pattern references and creating a database to store these relationships on the basis of standardized quality attribute definitions is valuable knowledge.

Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it's not difficult to use.

Requirements elicitation/modeling/analysis (N=15)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Documentation</th>
<th>Architecture</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Project Management</th>
<th>Requirements</th>
<th>Project Management</th>
<th>Requirements</th>
<th>Project Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>4.3</td>
<td>4.1</td>
<td>4.3</td>
<td>4.1</td>
<td>4.2</td>
<td>4.1</td>
<td>4.1</td>
<td>4.2</td>
<td>4.1</td>
<td>4.1</td>
</tr>
<tr>
<td>SD</td>
<td>.6</td>
<td>.9</td>
<td>.7</td>
<td>1.0</td>
<td>1.1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.2</td>
<td>1.0</td>
</tr>
<tr>
<td>p-value</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
</tbody>
</table>

Table D.2: Descriptive statistics and comparison of the population mean to a hypothesized neutral value of 3, using t-tests for each of the statement.
Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field. Identifying standard quality attribute definitions within current pattern references is critical for comparing the same patterns against the quality attribute they possess. Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge. Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.

Figure D.1: Stacked bar chart: Agreement with the four statements by different sectors.
Table D.4: Descriptive statistics and test of the population mean to a hypothesized neutral value at (3) using t-test for each of the subgroups with the
sectors of work experience.

<table>
<thead>
<tr>
<th>Sector</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government</td>
<td>4.0</td>
<td>1.2</td>
<td>0.001</td>
</tr>
<tr>
<td>Industry</td>
<td>3.7</td>
<td>0.9</td>
<td>0.003</td>
</tr>
<tr>
<td>Academia</td>
<td>4.1</td>
<td>0.9</td>
<td>0.002</td>
</tr>
</tbody>
</table>

---

**Table Key:**
- **Mean**
- **Standard Deviation**
- **p-value corresponding to a t-test with null hypothesis \( \mu \leq 3 \) and alternative hypothesis \( \mu > 3 \).**
However, when looking at the final statement, the highest mean comes from developers who gained experience in "government", followed by “industry” and “academia”. This is interesting, as academics agreed with the first three statements that identifying, studying, and creating the information that focuses on the problem between quality attributes and software patterns. However, they show much less support for developing an evaluation model to be used to solve the problem domain partially (last statement), which does not make sense. Developers from “industry”, however, agreed with the solution much more, which make reasonable agreement in pointing to the problem and producing a solution to fix it. We can explain these differences in general agreement by observing the differences between the natures of developers work in different sectors. Academics focus more on research and knowledge, whereas developers in industry aim to provide practical solutions. The support to produce a practical solution gives weight to our efforts doing this research. In general, all of the statements population mean for all sectors are significantly higher than 3 (neutral value) at 1% level of significance, except for government sector were not significantly agreed or disagreed to the statement “Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess”.

D.3.4 Analyses of (Q3 and Q4):

Figure D.2: Stacked bar chart: Years of experience in software development field with the developers awareness of software style/patterns.

D.3.5 Analyses of (Q4 and each of the questions from Q17 to Q20):

Table D.5: Percentage frequency table: Awareness of software style/patterns by developers.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Percentage</th>
<th>Valid Percentage</th>
<th>Cumulative Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>No</td>
<td>9</td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>41</td>
<td>78.8</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>50</td>
<td>96.2</td>
</tr>
<tr>
<td>Missing</td>
<td>System</td>
<td>2</td>
<td>3.8</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>52</td>
<td>100</td>
</tr>
</tbody>
</table>

41 respondents were “aware of software styles/patterns”, 9 were unaware and 2 did not respond. The relationship between developers who said they were “aware of software styles/patterns” and the four statements is shown in the (below).

The above bar chart shows the mean responses of the developers who said they were “aware of software style/patterns”. All statements had means significantly higher than 3 (the neutral value). The error bars (95% CI) show that all of the mean responses are not statistically different from each other,
as each error bar overlaps each of the others. (On the graph this is illustrated by the fact that they all contain "4").

D.3.6 Examples of 3 and 4 dimensional analysis:

Since the dataset contained only 52 observations (with 14 missing observations after Q8), three and four dimensional comparison matrices were futile, as with few observations in each category, Chi-square tests were invalid. Tables (D.6, and E.40), show (4 and 3) dimensions analysis examples, respectively.

Table D.6: ANOVA-1, Analysis of Question 1/ Option 1 “Requirements elicitation/modelling/analysis”, for Q17 to Q20 by Q2.

<table>
<thead>
<tr>
<th>Q1</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field.</td>
<td>Between Groups</td>
<td>.17</td>
<td>2</td>
<td>.08</td>
<td>.19</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>5.17</td>
<td>12</td>
<td>.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>5.33</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess.</td>
<td>Between Groups</td>
<td>.06</td>
<td>2</td>
<td>.03</td>
<td>.03</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>10.88</td>
<td>12</td>
<td>.91</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>10.93</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge.</td>
<td>Between Groups</td>
<td>1.39</td>
<td>2</td>
<td>.70</td>
<td>.88</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>9.54</td>
<td>12</td>
<td>.80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>10.93</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.</td>
<td>Between Groups</td>
<td>.23</td>
<td>2</td>
<td>.12</td>
<td>.17</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>8.17</td>
<td>12</td>
<td>.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>8.40</td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For developers whose general field of expertise is “Requirements elicitation/modelling/analysis”, there was no significant difference in responses to each of the four statements in relation work sectors, in which they gained their most experience.

322
Because of low frequency in four dimensional analyses, all the results are either invalid or non-significant. The following table shows the cross tabulation of (Q2 and Q3 with Q4 and Q5), where most of the cell counts are either 0 or 1. Therefore, four dimensional analyses for this data are invalid.

<table>
<thead>
<tr>
<th>How many years experience do you have in total in the software/systems development field?</th>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>Never</th>
<th>Infrequently (&lt; 10%)</th>
<th>Reasonably frequently (&gt; 15% and &lt; 50%)</th>
<th>Regularly (&gt; 50% and &lt; 80%)</th>
<th>Nearly always (&gt; 90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td>0mm aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>Yes, which language?</td>
<td>No</td>
<td>Yes, which language?</td>
<td>No</td>
<td>Yes, which language?</td>
<td>No</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
<td>Count</td>
<td>Count</td>
<td>Count</td>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>5–10 (years)</td>
<td>Academia</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Industry</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Government</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10–15 (years)</td>
<td>Academia</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Industry</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Government</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>15–20 (years)</td>
<td>Academia</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20–25 (years)</td>
<td>Academia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>Academia</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
D.4 Snapshots of primitive analysis database

Figure D.3: Overview page - for the primitive analysis database

Figure D.4: Responses primitive analysis page - Respondent’s countries, total answers, percentage of their answers out of 20 questions.
D.5. SUMMARY TABLES FOR MOST IMPORTANT RESULTS

Figure D.5: Selection criteria page - were one or more questions should be selected and submitted by clicking 'submit' button to get the analysis results.

D.5 Summary Tables for most important results

This section represents several tables that summarise valuable analyses results, to facilitate the readability, and to reduce unnecessary tables that were generated during the analysis process.

Table D.8: Cross tabulation and test of independence between how often developers used software style/patterns and whether the developers support standard documentation practices for software patterns.

<table>
<thead>
<tr>
<th>How often do you use software styles/patterns during your work?</th>
<th>Never</th>
<th>Infrequently (&lt;10%)</th>
<th>Reasonably frequently (&gt;15% and &lt;50%)</th>
<th>Regularly (&gt;50% and &lt;80%)</th>
<th>Nearly always (&gt;90%)</th>
<th>Chi-Square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support of standard documentation practices for software patterns</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>9.2 (.690)</td>
</tr>
<tr>
<td>Not sure</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Perhaps</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX D. COMPLEMENT INFORMATION FOR THE SPS SURVEY

Table D.9: Cross tabulation of how often developers used software style/patterns with the main factors that discourage and encourage the utilization of software patterns by developers and Chi-square test of independence between attributes.

<table>
<thead>
<tr>
<th>How often do you use software styles/patterns during your work?</th>
<th>Never</th>
<th>Infrequently (&lt;10%)</th>
<th>Reasonably frequently (&gt;15% and &lt;50%)</th>
<th>Regularly (&gt;50% and &lt;80%)</th>
<th>Nearly always (&gt;90%)</th>
<th>Chi-Square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The main factors that <strong>discourage</strong> the utilization of software patterns by developers.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No or few available references</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Poor documentation of existing software patterns</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Very little teaching of patterns in academic institutions or industry</td>
<td>47</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>28.2 (.659)</td>
</tr>
<tr>
<td>No proof of the solutions provided by patterns</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unknown quality attributes for combining software patterns</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Developing new solutions saves more time than searching for, and implementing the right patterns</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Hard to integrate with other components or existing systems</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Other – please specify:</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>The main factors that <strong>encourage</strong> the utilization of software patterns by developers.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find the right patterns that solve the problems encountered</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Most available references are clear and well documented</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Easy to implement</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Clear identification of quality attributes possessed by patterns</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Other - please specify:</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table D.10: Group descriptive statistics for each of the four statements in first column along with independent t-test results for testing difference between whether developers frequently used (≥ 3) software style/patterns during their work or not (< 3).

<table>
<thead>
<tr>
<th>How often do you use software styles / patterns during your work?</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>Levene’s Test for Equality of Variances: F (p-value)</th>
<th>t-test for Equality of Means</th>
<th>95% CI of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>t-statistic (p-value)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 3</td>
<td>17</td>
<td>4.06</td>
<td>.75</td>
<td>0.21</td>
<td>-0.49</td>
<td>(-0.60, 0.37)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>17</td>
<td>4.18</td>
<td>.64</td>
<td>(.650)</td>
<td>(.625)</td>
<td></td>
</tr>
<tr>
<td>Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 3</td>
<td>16</td>
<td>3.75</td>
<td>.78</td>
<td>2.05</td>
<td>0.39</td>
<td>(-0.53, 0.78)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>16</td>
<td>3.63</td>
<td>1.03</td>
<td>(.163)</td>
<td>(.700)</td>
<td></td>
</tr>
<tr>
<td>Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 3</td>
<td>16</td>
<td>4.19</td>
<td>.66</td>
<td>5.46</td>
<td>0.74</td>
<td>(-0.45, 0.95)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>16</td>
<td>3.94</td>
<td>1.18</td>
<td>(.026)</td>
<td>(.467)</td>
<td></td>
</tr>
<tr>
<td>Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥ 3</td>
<td>16</td>
<td>4.00</td>
<td>.73</td>
<td>2.30</td>
<td>0.63</td>
<td>(-0.46, 0.86)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>15</td>
<td>3.80</td>
<td>1.01</td>
<td>(.141)</td>
<td>(.532)</td>
<td></td>
</tr>
</tbody>
</table>
Table D.11: Cross tabulation of whether the relationship between the patterns and the quality attributes in those references been proved scientifically or otherwise with the main factors that discourage and encourage the utilisation of software patterns by developers and Chi-square test of independence between attributes.

<table>
<thead>
<tr>
<th>Has the relationship between the patterns and the quality attributes in those references been proved scientifically or otherwise?</th>
<th>No</th>
<th>Yes</th>
<th>Not sure</th>
<th>Perhaps</th>
<th>Chi-Square (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The main factors that discourage the utilization of software patterns by developers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19.43 (.366)</td>
</tr>
<tr>
<td>Poor documentation of existing software patterns</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Very little teaching of patterns in academic institutions or industry</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>No proof of the solutions provided by patterns</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Unknown quality attributes for combining software patterns</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Developing new solutions saves more time than searching for, and implementing the right patterns</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Hard to integrate with other components or existing systems</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>The main factors that encourage the utilization of software patterns by developers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.8 (.615)</td>
</tr>
<tr>
<td>Easy to find the right patterns that solve the problems encountered</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Most available references are clear and well documented</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Easy to implement</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Clear identification of quality attributes possessed by patterns</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Other – please specify</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Table D.12: F statistic (p-value) from analysis of variance (ANOVA) for Q17-Q20 by the sectors that developers gained most of their general software development experience (Q2)?

<table>
<thead>
<tr>
<th>Statements</th>
<th>Requirements elicitation/modelling/analysis</th>
<th>Project management</th>
<th>Architecture</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field.</td>
<td>0.19 (.827)</td>
<td>0.55 (.587)</td>
<td>0.40 (.679)</td>
<td>0.04 (.957)</td>
<td>0.08 (.923)</td>
<td>1.65 (.228)</td>
<td>0.25 (.787)</td>
</tr>
<tr>
<td>Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess.</td>
<td>0.03 (.968)</td>
<td>0.91 (.427)</td>
<td>2.64 (.116)</td>
<td>0.05 (.950)</td>
<td>.11 (.896)</td>
<td>0.20 (.821)</td>
<td>0.79 (.496)</td>
</tr>
<tr>
<td>Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge</td>
<td>0.88 (.442)</td>
<td>1.02 (.385)</td>
<td>0.39 (.684)</td>
<td>0.52 (.604)</td>
<td>0.37 (.692)</td>
<td>1.38 (.283)</td>
<td>0.50 (.632)</td>
</tr>
<tr>
<td>Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.</td>
<td>0.17 (.844)</td>
<td>0.03 (.967)</td>
<td>1.39 (.289)</td>
<td>0.30 (.747)</td>
<td>0.47 (.635)</td>
<td>0.48 (.631)</td>
<td>1.19 (.368)</td>
</tr>
</tbody>
</table>
Table D.13: F statistic (p-value) from analysis of variance (ANOVA) for Q17-Q20 by the developer’s total years of experience in the software development field (Q3).

<table>
<thead>
<tr>
<th>Statements</th>
<th>Requirements elicitation modelling analysis</th>
<th>Project management</th>
<th>Architecture</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying the relationship between software patterns and quality attributes is very important to software developers and the software engineering field.</td>
<td>0.12 (.947)</td>
<td>0.15 (.959)</td>
<td>0.42 (.792)</td>
<td>0.44 (.781)</td>
<td>0.60 (.670)</td>
<td>0.08 (.972)</td>
<td>0.03 (.991)</td>
</tr>
<tr>
<td>Identifying standard quality attribute definitions within current pattern references is a critical for comparing the same patterns against the quality attribute they possess.</td>
<td>0.15 (.927)</td>
<td>0.78 (.557)</td>
<td>0.06 (.993)</td>
<td>0.29 (.881)</td>
<td>.82 (.530)</td>
<td>0.35 (.790)</td>
<td>0.40 (.758)</td>
</tr>
<tr>
<td>Studying relationships between patterns and quality attributes based on the current reliable software pattern references, and creating a database to store these relationships on the basis of standardized quality attribute definitions, is valuable knowledge.</td>
<td>0.71 (.568)</td>
<td>0.75 (.578)</td>
<td>0.28 (.887)</td>
<td>0.40 (.804)</td>
<td>0.73 (.583)</td>
<td>0.22 (.883)</td>
<td>0.36 (.785)</td>
</tr>
<tr>
<td>Developing an evaluation model to assess patterns against quality attributes is worthwhile, provided it’s not difficult to use.</td>
<td>0.41 (.749)</td>
<td>0.57 (.690)</td>
<td>0.52 (.723)</td>
<td>0.88 (.499)</td>
<td>1.25 (.326)</td>
<td>0.38 (.773)</td>
<td>0.62 (.630)</td>
</tr>
</tbody>
</table>
Complement information for SA Survey

This appendix represents supportive materials for Chapter 5, where tables, figures, and sections are ordered based on the description flow within the main chapter, in order to facilitate their traceability.

E.1 Survey Questions

Summary of the survey questions objectives, are shown in Table E.1.

Table E.1: Summary of the questionnaire and each section objectives.

<table>
<thead>
<tr>
<th>Section number</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This section (Q1 to Q5) is for collecting demographical data about the participants, same as the case in Chapter 4.</td>
</tr>
<tr>
<td>2</td>
<td>The second section consisted of five questions (Q6 to Q10). These questions mainly focused on the matters relating to SA utilisation, description, and modelling amongst software developers. Also, it includes the factors that are likely have an affect on the utilisation of SA modelling techniques during development process. The section went further to gather information related to software description languages.</td>
</tr>
<tr>
<td>3</td>
<td>The final section consists of thirteen questions (Q11-Q23), which focused on SAE tactics, and factors that could support or hinder SAE methods. The section also, sought to explore the effect of the current technologies, automation and tools on SAE methodologies.</td>
</tr>
</tbody>
</table>

E.2 Individual Analysis (One dimension _ Descriptive statistics)

This section represents important data that could support individual analysis discussion, which is reported in Chapter 5.
E.2.1 Analyses of (Q8)

According to more than half of the respondents, Semi-formal language and Natural language used together form the best language to describe software/system architecture. A substantial number of respondents also seemed comfortable with using semi-formal languages such as UML as the best language to use in software/system description. This information can be useful to determine which languages need to be given priority in documentation and improvement in order to improve the use of models in describing system/software architecture.

Figure E.1: Bar chart for the best language to use to describe software/system architecture as identified by the respondents.

E.2.2 Analyses of (Q9 and Q10)

There is an agreement of the respondents that developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and re-usability of the resulting architecture. On the other hand, where the respondents have taken a neutral position about the fact that usage of software style/pattern concepts & models during architecture development increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation. However, the median value for the two items are equal to 3, which signifies respondents may have a neutral position for both items. The analysis results are shown in table E.2. Figure E.2 indicates that there is a tendency towards agreement for the first item, whereas for the second item there is neither general agreement nor disagreement.

Figure E.2: Box plot of two items Q9 and Q10.

A One-sample t-test for specified mean value of 3 confirms the results above with t = 4.1, p-value < .01 for the first item.
Table E.2: Descriptive statistics Q9 and Q10.

<table>
<thead>
<tr>
<th></th>
<th>Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture.</th>
<th>Usage of software style/pattern concepts models during architecture development, increases the utilisation of modelling description languages, <strong>BUT</strong> decreases the simplicity of the architecture valuation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N Valid</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>Missing</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Mean</td>
<td>3.5</td>
<td>3.0</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>.8</td>
<td>.8</td>
</tr>
<tr>
<td>Skewness</td>
<td>.32</td>
<td>.0</td>
</tr>
<tr>
<td>Std. Error of Skewness</td>
<td>.54</td>
<td>.4</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>-.2</td>
<td>-.4</td>
</tr>
<tr>
<td>Std. Error of Kurtosis</td>
<td>.7</td>
<td>.7</td>
</tr>
<tr>
<td>Percentiles 25</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>50</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>75</td>
<td>4.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

**E.2.3 Analyses of (Q12):** "Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?"

The percentage of respondents who neither know nor use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics is very large. This is an indication that there is still a lot that needs to be done to increase awareness and encourage the use of architectural evaluation methods.

![Pie Chart](image)

Figure E.3: Pie Chart of the percentage of the respondents who know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics.
E.3 Two dimensions matrices analysis

This section represents all necessary information that support the argument reported in Chapter 2 concerning 2-dimensions analyses.

E.3.1 Information related to the significant results

E.3.1.1 Analyses of Q1 with (Q9 and Q10)

An independent sample t-test was used to test whether two group means corresponding to the respondents with more experience in “Architecture”, and those with other expertise are equal for the statements in (Q9 and Q10). The results are shown in Table E.3.

According to the results, the group mean for the item “Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and re-usability of the resulting architecture” corresponding to the developers whose general field of expertise regarding software development is “Architecture” is significantly higher than the rest of the developers with other expertise, $t=1.90$, p-value < .05. Thus, architects tend to agree more with the statement. There is no significant difference in means for the second statement, $t=-1.07$, p-value > .05.

Table E.3: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Architecture” and those have other expertise.

| What is your general field of expertise regarding software development? | Levene's Test for Equality of Variances | t-test for Equality of Means |
|---|---|---|---|---|---|---|---|---|
| | F | Sig. | t | df | Sig. (One-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture. | Equal variances assumed | 2.69 | .109 | 1.90 | 43 | .032 | .43 | .22 | -.03 | .88 |
### E.3.1.2 Analyses of Q1 with (Q15–Q23)

Table E.4: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Project Management” and those who have expertise in other fields apart from this.

<table>
<thead>
<tr>
<th>What is your general field of expertise regarding software development?</th>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer: Project Management.</td>
<td>Equal variances assumed</td>
<td>Equal variances not assumed</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>Sig.</td>
</tr>
<tr>
<td>Most of the existing software architecture evaluation methods, produce qualitative results.</td>
<td>0.02</td>
<td>.895</td>
</tr>
</tbody>
</table>

Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, *BUT* decreases the simplicity of the architecture valuation.

<table>
<thead>
<tr>
<th>Equal variances assumed</th>
<th>Equal variances not assumed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.77</td>
<td>27.4</td>
</tr>
<tr>
<td>.74</td>
<td>.393</td>
</tr>
<tr>
<td>-1.02</td>
<td>32.7</td>
</tr>
</tbody>
</table>
Among all the 9 items, the only significant difference in means was noted in the item “Most of the existing software architecture evaluation methods, produce qualitative results”, $t = -2.35$, $p$-value $< 0.05$. It is noted that for project managers, the population mean for this item is significantly less than that of the respondents with other expertise. The mean difference is insignificant for all the other 8 items.

Table E.5: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Design” and those who have expertise in other fields apart from this.

| What is your general field of expertise regarding software development? Answer: Design | Levene’s Test for Equality of Variances | t-test for Equality of Means | 95% Confidence Interval of the Difference |
|---|---|---|---|---|---|---|
| | $F$ | Sig. | $t$ | df | Sig. (One-tailed) | Mean Difference | Std. Error Difference | Lower | Upper |
| Most of the existing software architecture evaluation methods, produce qualitative results | Equal variances assumed | 0.29 | .595 | -2.10 | 42 | .021 | -0.54 | 0.26 | -1.05 | -0.02 |
| | Equal variances not assumed | | 0.04 | 36.1 | 0.20 | -0.54 | 0.25 | -1.04 | -0.03 |
| Reliable tools are important for developing/or evaluating software/system architectures. | Equal variances assumed | 3.49 | .069 | 2.04 | 42 | .024 | 0.37 | 0.18 | 0.00 | 0.73 |
| | Equal variances not assumed | | 2.04 | 31.3 | 0.25 | 0.37 | 0.18 | 0.00 | 0.73 |

The next field of expertise to be tested for the 9 items was Design. Table E.5 shows the detailed results. Two significant results were noted: First, the group...
means for the item “Most of the existing software architecture evaluation methods, produce qualitative results” showed a significant difference for the expertise in Design, with \( t = -2.10, \) \( p\text{-value} < 0.05 \). Thus for this group, the response to the item depended on the field of expertise. Secondly, there is also a significant difference between the group means of Designers and respondents in other fields of expertise for the item “Reliable tools are important for developing/evaluating software/system architectures \( ^{*} \), \( t = 2.04, \) \( p\text{-value} < 0.05 \). Thus, the population mean for this item is significantly greater for designers compared to respondents with other expertise.

The rest of the seven items have insignificant mean difference for the two groups, with the \( p\text{-value} \) for all the tests being greater than alpha value of .05.

Table E.6: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Architecture” and those who have expertise in other fields apart from this.

<table>
<thead>
<tr>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Sig.</td>
<td>t</td>
</tr>
<tr>
<td>Equal variances assumed</td>
<td>0.06</td>
<td>.806</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>2.24</td>
<td>.016</td>
</tr>
</tbody>
</table>

The next expertise under consideration for the 9 items was “Architecture”. For this field of expertise, the full analysis is shown in table E.6.

Only the group means for the item “Architecture is design, but NOT all design is Architecture”, show a significant difference, \( t = 2.20, \) \( p\text{-value} < 0.05 \). The \( t \) value shows that the population mean is actually much greater. The means are insignificant for the rest of the 8 items.

Table E.7: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Coding” and those who have expertise in other fields apart from this.

<table>
<thead>
<tr>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Sig.</td>
</tr>
<tr>
<td>Equal variances assumed</td>
<td></td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td></td>
</tr>
</tbody>
</table>
According to the results displayed in Table E.7, the population mean of the responses for the item “Architecture is design, but *NOT* all design is Architecture” corresponding to the respondents whose general field of expertise regarding software development is “Coding” is significantly less than that of the respondents in other fields of expertise, $t=-2.35$, p-value=.012 < .05.

The other significant result for coding experts was deduced for the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”, in which the population mean for the same field of expertise is significantly less than other fields of expertise.
expertise, $t = -2.50$, p-value$= .009 < .01$. No significant result was deduced for the rest of the six items for the field of expertise of “coding”, the p-value for all the tests are greater than alpha value of .05.

Table E.8: Equality of means between two groups corresponding to the respondents whose general field of expertise regarding software development is “Testing” and those who have expertise in other fields apart from this.

<table>
<thead>
<tr>
<th>What is your general field of expertise regarding software development?</th>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer: Testing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equal variances assumed</th>
<th>F</th>
<th>Sig.</th>
<th>t</th>
<th>df</th>
<th>Sig. (One-tailed)</th>
<th>Mean Difference</th>
<th>Std. Error Difference</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Architecture is design, but NOT all design is Architecture&quot;</td>
<td>0.02</td>
<td>.878</td>
<td>-2.09</td>
<td>43</td>
<td>.021</td>
<td>-0.53</td>
<td>0.26</td>
<td>-1.05</td>
<td>-0.02</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>-2.10</td>
<td>31.3</td>
<td>.022</td>
<td>-0.53</td>
<td>0.25</td>
<td>-1.05</td>
<td>-0.01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes. | Equal variances assumed | 6.89 | .012 | -1.58 | 42 | .61 | -0.38 | 0.24 | -0.87 | 0.11 |
| Equal variances not assumed | -1.79 | 41.6 | .041 | -0.38 | 0.21 | -0.82 | 0.05 |

Table E.8 shows the analysis of the population means for the responses to the 9 items for the respondents whose field of expertise is “Testing”. For this expertise, the population mean of the responses for the item “Architecture is design, but NOT all design is Architecture” is significantly less compared to those of other fields of expertise, $t = -2.09$, p-value$= .021 < .05$.

Further, the population mean of the responses for the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes” for this expertise is significantly less than that of the respondents whose have other expertise, $t = -1.79$, p-value$= .041 < .05$.

The other seven items for the field of expertise of “Testing” had a p-value greater than alpha value of .05, thus the results were insignificant.
E.3.1.3 Analyses of Q2 with (Q9 and Q10)

A one way ANOVA analysis was conducted to see whether group means corresponding to the categories of a variable are different for the variable of interest. The full analysis has been tabulated in Table E.9.

According to the results, Group means of the item “Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture” corresponding to the sector in which developers gained most of their general software development experience are not significantly different, $F = .01$, $p$-value $>.05$. This shows that the responses to the item are not significantly affected by the three sectors in which respondents gained their general software development experience.

On the other hand, the responses to the item “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, **BUT** decreases the simplicity of the architecture valuation” corresponding to the item “In which of the following sectors have you gained most of your general software development experience?” are significantly different, $F = 3.33$, $p$-value $< .05$, showing that the responses to the items have a significant variation with the three sectors in which developers gained most of their software development experience.

Since the group means are significantly different for these items, there was a need to confirm which pairs of means are significantly different to each other, arranged according to the post hoc test (LSD method).

Table E.9: One way ANOVA table to test the difference between two variables of interest for Q9 and Q10.

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture.</td>
<td>Between Groups</td>
<td>.01</td>
<td>2</td>
<td>.01</td>
<td>.990</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>25.19</td>
<td>42</td>
<td>.60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>25.20</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, <strong>BUT</strong> decreases the simplicity of the architecture valuation.</td>
<td>Between Groups</td>
<td>4.29</td>
<td>2</td>
<td>2.15</td>
<td>.333</td>
</tr>
<tr>
<td></td>
<td>Within Groups</td>
<td>27.71</td>
<td>43</td>
<td>.64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>32.00</td>
<td>45</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to the analysis shown in Table E.10, the group mean for “Academia” is significantly higher than the group mean for “Industry” for the item “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, **BUT** decreases the simplicity of the architecture valuation.”, Mean difference $= 0.63$, $p$-value $< .05$. Other group means are not significantly different.
Table E.10: Multiple comparison test for the item in Q10, according to the the group means for the three sectors in which developers gained their software development experience using LSD method.

<table>
<thead>
<tr>
<th>Dependent Variable</th>
<th>(I) In which of the following sectors have you gained most of your general software development experience?</th>
<th>(J) Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, <strong>BUT</strong> decreases the simplicity of the architecture valuation.</th>
<th>Mean Difference (I-J)</th>
<th>Std. Error</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Academia</td>
<td>Industry</td>
<td>.63*</td>
<td>.26</td>
<td>.021</td>
</tr>
<tr>
<td></td>
<td>Academy</td>
<td>Government</td>
<td>.57</td>
<td>.33</td>
<td>.094</td>
</tr>
<tr>
<td></td>
<td>Industry</td>
<td>Academia</td>
<td>-.63*</td>
<td>.26</td>
<td>.021</td>
</tr>
<tr>
<td></td>
<td>Industry</td>
<td>Government</td>
<td>-.06</td>
<td>.35</td>
<td>.858</td>
</tr>
<tr>
<td></td>
<td>Government</td>
<td>Academia</td>
<td>-.57</td>
<td>.33</td>
<td>.094</td>
</tr>
<tr>
<td></td>
<td>Industry</td>
<td>Government</td>
<td>.06</td>
<td>.35</td>
<td>.858</td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.

Table E.11: Cross tabulation of the sectors that developer’s gained most of their general software development experiences against selected categorical questions and corresponding results for Chi-square test of independence.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>χ² (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academia</td>
<td>Industry</td>
</tr>
<tr>
<td>Are you aware of any software/system architectural description/modelling languages?</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>19</td>
</tr>
<tr>
<td>No</td>
<td>5</td>
</tr>
</tbody>
</table>

| How often do you use models to describe software/system architecture during your work?           |             |            |
| Never                                                                                            | 3           | 2          | 2          |        |        |
| Infrequently (<10%)                                                                               | 12          | 3          | 3          | 9.64   | (.291) |
| Reasonably frequent (>15% and <50%)                                                              | 5           | 8          | 3          |        |        |
| Regularly (>50% and <80%)                                                                         | 3           | 2          | 0          |        |        |
| Nearly always (>90%)                                                                              | 1           | 3          | 0          |        |        |

<table>
<thead>
<tr>
<th>What are the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
</tr>
</tbody>
</table>
Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily  & 2 & 3 & 3 \\
It makes the designers/programers job much easier & 10 & 3 & 1 \\
Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development life cycle & 6 & 4 & 1 \\
Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier & 3 & 1 & 0 \\
The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible & 1 & 1 & 0 \\
Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g., SysML, \( \frac{1}{2} \) UML) & 2 & 3 & 1 \\
Teaching of the architecture modelling languages in the academic sectors & 2 & 4 & 2 \\

What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture?  &  &  &  \\
Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system & 10 & 6 & 3 \\
Lack of standardization between existing architecture modelling technique, notations, and semantics & 1 & 6 & 1 \\
Current architecture description languages (including modelling languages) are still immature & 4 & 3 & 1 \\
Modelling the architecture has limited benefit to the whole software/system development process, so it's to some extent a waste of time and money & 4 & 5 & 3 \\
Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance) & 6 & 2 & 5 \\

From your experience, what is the best language to use to describe software/system architecture, so as to be more useful to all stakeholders, and to be easier to undertake qualitative and quantitative assessments?  &  &  &  \\
Natural language only & 2 & 1 & 1 \\
Semi-formal language & 5 & 2 & 1 \\
Formal language only & 1 & 0 & 0 \\
Formal language & Natural language together & 1 & 1 & 0 \\

11.46 (.323)  

6.44 (.893)
E.3. TWO DIMENSIONS MATRICES ANALYSIS

| Semi-formal language & Natural language together | 9 | 9 | 6 |
| Semi-formal language & Formal language together | 1 | 0 | 0 |
| All three languages above together | 3 | 3 | 0 |

Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models

| Yes | 1 | 5 | 1 | 5.43 (.066) |
| No | 22 | 11 | 7 |

Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?

| Yes | 2 | 2 | 1 | 0.23 (.892) |
| No | 21 | 13 | 7 |

What are the most important factors that could SUPPORT quantitative evaluation for any software architecture (SA)?

| The language used for describing SA | 4 | 2 | 1 | 7.68 (.660) |
| Formality level of SA description | 7 | 4 | 1 |
| Using standard language and architecture framework for describing SA | 11 | 5 | 4 |
| Tools availability for describing and evaluating SA | 8 | 10 | 4 |
| Documenting mechanism used during SA description | 6 | 2 | 1 |

What are the most important factors that could HINDER quantitative evaluation for any SA?

| The language used for describing SA | 10 | 4 | 3 | 8.93 (.539) |
| Formality level of SA description | 9 | 8 | 4 |
| Using standard language and architecture framework for describing SA | 4 | 0 | 1 |
| Tools availability for describing and evaluating SA | 8 | 6 | 1 |
| Documenting mechanism used during SA description | 3 | 1 | 2 |

E.3.1.4 Analyses of (Q3 and Q17)

The group means of the item “Most of the existing software architecture evaluation methods, produce qualitative results” corresponding to the categories of the respondents years’ experience in total in the software/systems development field are significantly different, $F = 3.76$, p-value $< .05$. This is evidence of existence of a significant relationship between the two items.

Table E.12: One way ANOVA analysis for the item “Most of the existing software architecture evaluation methods, produce qualitative results” corresponding to the categories of the respondent’s years of experience in total in the software/systems development field.

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>8.779</td>
<td>4</td>
<td>2.195</td>
<td>3.760</td>
<td>.011</td>
</tr>
<tr>
<td>Within Groups</td>
<td>22.767</td>
<td>39</td>
<td>.584</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>31.545</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Since the ANOVA analysis confirms that the group means are not all simultaneously equal, Tukey’s HSD post hoc test for multiple comparisons was used to test which pair of group mean are significantly unequal. The analysis is shown in Table E.13.

The group mean for the respondents who have 5-10 years of experience is significantly higher than the group mean for the respondents who have over 25 years of experience for the item “Most of the existing software architecture evaluation methods, produce qualitative results”, Mean difference = 1.33, p-value = 0.011 < .05. This shows that the respondents who have 5-10 years of experience in software development field have higher agreement to the item than the respondents who have over 25 years of experience.

Furthermore, the group mean for the respondents who have 10-15 years of experience is significantly higher than the group mean for the respondents who have over 25 years of experience for the item “Most of the existing software architecture evaluation methods, produce qualitative results”, Mean difference = 1.70, p-value = 0.003 < .01. Thus, the respondents who have 10-15 years of experience in software development field have higher agreement to the item than the respondents who have over 25 years of experience.

**Multiple Comparisons**

*Most of the existing software architecture evaluation methods, produce qualitative results Tukey’s HSD.*

Table E.13: Post hoc test (multiple comparisons) result using Tukey’s HSD test.

<table>
<thead>
<tr>
<th>(I) How many years’ experience do you have in total in the software/systems development field?</th>
<th>(J) How many years’ experience do you have in total in the software/systems development field?</th>
<th>Mean Difference (I-J)</th>
<th>Std. Error</th>
<th>Sig.</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 - 10 (years)</td>
<td>10 - 15 (years)</td>
<td>-.37</td>
<td>.29</td>
<td>.723</td>
<td>-1.21</td>
</tr>
<tr>
<td>15 - 20 (years)</td>
<td>-.33</td>
<td>.47</td>
<td>.954</td>
<td>-1.68</td>
<td>1.02</td>
</tr>
<tr>
<td>20 - 25 (years)</td>
<td>.00</td>
<td>.35</td>
<td>1.000</td>
<td>-1.01</td>
<td>1.01</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>1.33*</td>
<td>.42</td>
<td>.022</td>
<td>.14</td>
<td>2.53</td>
</tr>
<tr>
<td>10 - 15 (years)</td>
<td>5 - 10 (years)</td>
<td>.37</td>
<td>.29</td>
<td>.723</td>
<td>-.47</td>
</tr>
<tr>
<td>15 - 20 (years)</td>
<td>.03</td>
<td>.50</td>
<td>1.000</td>
<td>-1.40</td>
<td>1.47</td>
</tr>
<tr>
<td>20 - 25 (years)</td>
<td>.37</td>
<td>.40</td>
<td>.884</td>
<td>-.76</td>
<td>1.49</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>1.70*</td>
<td>.45</td>
<td>.005</td>
<td>.41</td>
<td>2.99</td>
</tr>
<tr>
<td>15 - 20 (years)</td>
<td>5 - 10 (years)</td>
<td>.33</td>
<td>.47</td>
<td>.954</td>
<td>-1.02</td>
</tr>
<tr>
<td>10 - 15 (years)</td>
<td>-.03</td>
<td>.50</td>
<td>1.000</td>
<td>-1.47</td>
<td>1.40</td>
</tr>
<tr>
<td>20 - 25 (years)</td>
<td>.33</td>
<td>.54</td>
<td>.972</td>
<td>-1.21</td>
<td>1.88</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>1.67</td>
<td>.58</td>
<td>.050</td>
<td>.00</td>
<td>3.34</td>
</tr>
<tr>
<td>20 - 25 (years)</td>
<td>5 - 10 (years)</td>
<td>.00</td>
<td>.35</td>
<td>1.000</td>
<td>-1.01</td>
</tr>
</tbody>
</table>
### E.3. TWO DIMENSIONS MATRICES ANALYSIS

<table>
<thead>
<tr>
<th></th>
<th>10 - 15 (years)</th>
<th>15 - 20 (years)</th>
<th>Over 25 (years)</th>
<th>10 - 15 (years)</th>
<th>15 - 20 (years)</th>
<th>Over 25 (years)</th>
<th>10 - 15 (years)</th>
<th>15 - 20 (years)</th>
<th>Over 25 (years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 - 15 (years)</td>
<td>-.37</td>
<td>.40</td>
<td>.884</td>
<td>-1.49</td>
<td>.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 - 20 (years)</td>
<td>-.33</td>
<td>.54</td>
<td>.972</td>
<td>-1.88</td>
<td>1.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>1.33</td>
<td>.49</td>
<td>.072</td>
<td>-0.08</td>
<td>2.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 - 10 (years)</td>
<td>-1.33*</td>
<td>.42</td>
<td>.022</td>
<td>-2.53</td>
<td>-.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 - 15 (years)</td>
<td>-1.70*</td>
<td>.45</td>
<td>.005</td>
<td>-2.99</td>
<td>-.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 - 20 (years)</td>
<td>-1.67</td>
<td>.58</td>
<td>.050</td>
<td>-3.34</td>
<td>.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 - 25 (years)</td>
<td>-1.33</td>
<td>.49</td>
<td>.072</td>
<td>-2.74</td>
<td>.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.
### E.3.1.5 Analyses of Q4 with (Q21 and Q23)

To test whether the item means corresponding to the respondents who are aware of any software/system architectural description/modelling languages was higher than the respondents who were not aware, an independent sample t-test was used. Table E.14 shows the results of the independent sample test.

The test yielded significant results for the item “Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task”, $t = 1.9$, p-value $= 0.04 < .05$. This means that the respondents who were aware of any software/system architectural description/modelling languages showed higher agreement to the above statement than those who were not aware of any SA description/modelling languages.

Table E.14: Independent sample t-test results between Q4 and the statements in (Q21 and Q23).

<table>
<thead>
<tr>
<th>Are you aware of any software/system architectural description/modelling languages, (e.g. ADLs, AADL, SysML, UML)?</th>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1: “Yes, which language?” Group 2: “No”</td>
<td>F</td>
<td>Sig.</td>
<td>t</td>
</tr>
<tr>
<td>Reading software/system architecture description models for automated evaluation purposes, is a critical, difficult, and error prone task.</td>
<td>Equal variances assumed</td>
<td>5.7</td>
<td>.02</td>
</tr>
<tr>
<td></td>
<td>Equal variances not assumed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes.</td>
<td>Equal variances assumed</td>
<td>4.6</td>
<td>.04</td>
</tr>
<tr>
<td></td>
<td>Equal variances not assumed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The test was also significant for the item “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”, $t = 2.1$, p-value $= 0.03 < .05$, implying that the respondents who were aware of any software/system architectural description/modelling languages showed higher agreement to the above statement than those who were not aware.

None of the other statements showed any significant mean difference for the respondents for this test.
E.3. TWO DIMENSIONS MATRICES ANALYSIS

E.3.1.6 Analyses of (Q5 and Q16):

A one way ANOVA procedure is used to test the difference among group means for the selected items corresponding to the categories of how often respondents used models to describe software/system architecture during their work. Table E.15 shows the results of the test.

The test is significant for the item “There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers”, $F=2.7$, $p$-value $<.05$. The difference between the group means corresponding to the categories based on how often respondents used models to describe software/system architecture during their work for the item “There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers” is an indication that the two are related.

Table E.15: ANOVA procedure for testing equality of group means for the selected item corresponding to the categories of how often respondents used models to describe software/system architecture during their work.

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>3.3</td>
<td>4</td>
<td>.8</td>
<td>2.7</td>
<td>.042</td>
</tr>
<tr>
<td>Within Groups</td>
<td>11.9</td>
<td>40</td>
<td>.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>15.2</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Further, to determine which group means are statistically different, a multiple comparison test (LSD method) was used. The results of the test are shown in Table E.16.

The group means for the item “There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers” are significantly unequal for the respondents who “Nearly always (>90%)” used models to describe software/system architecture during your work compared those who “Never” use SA at all, Mean difference $= 1.2$, $p$-value $< .01$.

Multiple Comparisons

There is still vagueness in the current literature concerning the differences between the architecture abstraction and high level design, which causes some confusion and perhaps wastes time during development by architects and designers. LSD.

Table E.16: Multiple comparison test results using LSD method.

<table>
<thead>
<tr>
<th>(I) How often do you use models to describe software/system architecture during your work?</th>
<th>(J) How often do you use models to describe software/system architecture during your work?</th>
<th>Mean Difference (I-J)</th>
<th>Std. Error</th>
<th>Sig.</th>
<th>95% Confidence Interval</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never Infrequently (&lt;10 %)</td>
<td></td>
<td>-.5</td>
<td>.3</td>
<td>.08</td>
<td>-1.0</td>
<td>-1.0</td>
<td>.1</td>
</tr>
<tr>
<td>Reasonably frequent (&gt;15 % and &lt;50 %)</td>
<td></td>
<td>-.5</td>
<td>.3</td>
<td>.08</td>
<td>-1.0</td>
<td>-1.0</td>
<td>.1</td>
</tr>
<tr>
<td>Infrequently (&lt;10 %)</td>
<td>Never</td>
<td>Regularly (&gt;50 % and &lt;80 %)</td>
<td>Nearly always (&gt;90 %)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>-----------------------------</td>
<td>-----------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fairly frequent (&gt;50 % and &lt;80 %)</td>
<td>.5</td>
<td>.3</td>
<td>.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearly always (&gt;90 %)</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nearly always (&gt;90 %)</th>
<th>Never</th>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Fairly frequent (&gt;50 % and &lt;80 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Regularly (&gt;50 % and &lt;80 %)</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Nearly always (&gt;90 %)</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Never</th>
<th>Fairly frequent (&gt;50 % and &lt;80 %)</th>
<th>Nearly always (&gt;90 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>.5</td>
<td>.3</td>
<td>.08</td>
</tr>
<tr>
<td>Regularly (&gt;50 % and &lt;80 %)</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Nearly always (&gt;90 %)</td>
<td>.0</td>
<td>.3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.

The group means are also unequal for the respondents who “Nearly always (>90 %)” used models to describe software/system architecture during their work as compared to those who “Infrequently (<10 %)” used, Mean difference = 0.7, p-value < .05.

Finally, the group means are unequal for the respondents who “Nearly always (>90 %)” used models to describe software/system architecture during their work compared to those who “Reasonable frequent (>15 % and <50 %)” used SA in their work, Mean difference = 0.7, p-value < .05.

E.3.1.7 Analyses of (Q5 and Q18):

The one way ANOVA procedure was also used to test the difference among group means for the selected item corresponding to the categories of how often respondents used models to describe soft-
The test is significant for the item “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures”, \( F = 2.7 \), \( p \)-value < .05. This means that the group means corresponding to the categories of how often respondents used models to describe software/system architecture during their work are statistically different for this item.

In order to determine the specific group means that were statistically different, a multiple comparison test (LSD method) is used as shown in Table E.18.

The group means for the item “It’s worthwhile to undertake an effort to develop a quantitative methodology for evaluating software/system architectures” showed a significant difference for the respondents who “Nearly always (>90%)” used models to describe software/system architecture during their work compared to those who “Never” used, Mean difference = 1.6, \( p \)-value < .01. Furthermore, the group means are unequal for the respondents who “Nearly always (>90%)” used models to describe software/system architecture during their work compared to those who “Infrequently (<10%)” used models in their work, Mean Mean difference = 1.2, \( p \)-value < .05.

Table E.17: ANOVA procedure for testing equality of group means for the selected item corresponding to the categories of how often respondents used models to describe software/system architecture during their work.

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>( F )</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>6.4</td>
<td>4</td>
<td>1.6</td>
<td>3.0</td>
<td>.03</td>
</tr>
<tr>
<td>Within Groups</td>
<td>20.7</td>
<td>38</td>
<td>.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>27.1</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The other difference in group means was noticed between the respondents who “Nearly always (>90%)” used models to describe software/system architecture during their work and those who “Reasonably frequent (>15% and <50%)” used the same, Mean difference = 1.0, \( p \)-value < .05. Finally, the group means were also unequal between the respondents who “Nearly always (>90%)” used models to describe software/system architecture during their work and those who “Regularly (>50% and <80%)” used models to describe SA during their work, Mean difference = 1.2, \( p \)-value < .05.

Multiple Comparisons

Table E.18: Multiple comparison test results using LSD method.

<table>
<thead>
<tr>
<th>(I) How often do you use models to describe software/system architecture during your work?</th>
<th>(J) How often do you use models to describe software/system architecture during your work?</th>
<th>Mean Difference (I-J)</th>
<th>Std. Error</th>
<th>Sig.</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>Infrequently (&lt;10 %)</td>
<td>-.4</td>
<td>.4</td>
<td>.23</td>
<td>-1.2</td>
</tr>
<tr>
<td>Reasonably frequent (&lt;15% and &lt;50 %)</td>
<td>-6</td>
<td>.4</td>
<td>.11</td>
<td>-1.3</td>
<td>.1</td>
</tr>
</tbody>
</table>
### APPENDIX E. COMPLEMENT INFORMATION FOR SA SURVEY

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Nearly always (&gt;90 %)</th>
<th>Infrequently (&lt;10 %)</th>
<th>Reasonably frequent (&gt;15 % and &lt;50 %)</th>
<th>Never</th>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Nearly always (&gt;90 %)</th>
<th>Never</th>
<th>Infrequently (&lt;10 %)</th>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Nearly always (&gt;90 %)</th>
<th>Never</th>
<th>Infrequently (&lt;10 %)</th>
<th>Regularly (&gt;50 % and &lt;80 %)</th>
<th>Nearly always (&gt;90 %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-.4</td>
<td>.4</td>
<td>.34</td>
<td>-.13</td>
<td>.5</td>
<td>.4</td>
<td>.23</td>
<td>-.3</td>
<td>.4</td>
<td>.3</td>
<td>.55</td>
<td>-.7</td>
<td>.4</td>
<td>-.2</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>-1.6*</td>
<td>.5</td>
<td>.00</td>
<td>-.25</td>
<td>-.6</td>
<td>-1.2*</td>
<td>.01</td>
<td>-.2</td>
<td>.6</td>
<td>.4</td>
<td>.67</td>
<td>-.6</td>
<td>1.0</td>
<td>.2</td>
<td>.4</td>
</tr>
<tr>
<td></td>
<td>.6</td>
<td>.4</td>
<td>.11</td>
<td>-.1</td>
<td>1.3</td>
<td>.2</td>
<td>.3</td>
<td>.55</td>
<td>-.4</td>
<td>.7</td>
<td>.6</td>
<td>.7</td>
<td>.4</td>
<td>.2</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>.4</td>
<td>.4</td>
<td>.34</td>
<td>-.5</td>
<td>1.3</td>
<td>.2</td>
<td>.4</td>
<td>.67</td>
<td>-.6</td>
<td>.6</td>
<td>.6</td>
<td>.6</td>
<td>.4</td>
<td>.2</td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td>.2</td>
<td>.5</td>
<td>.03</td>
<td>-.22</td>
<td>-.1</td>
<td>.2</td>
<td>.5</td>
<td>.03</td>
<td>.1</td>
<td>.1</td>
<td>-.2</td>
<td>.2</td>
<td>.2</td>
<td>.1</td>
<td>.2</td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.

#### E.3.1.8 Analyses of (Q5 and Q22)

The results of a one way ANOVA procedure for Q22 is shown in Table E.19. The test showed significant results for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier”, F = 3.0, p-value < .05. Therefore, the group means for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier” are affected by how often respondents used models to describe software/system architecture during their work.
Table E.19: ANOVA procedure for testing equality of group means for the selected item corresponding to the categories of how often respondents used models to describe software/system architecture during their work.

*Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier.*

<table>
<thead>
<tr>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>4.9</td>
<td>4</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Within Groups</td>
<td>16.1</td>
<td>39</td>
<td>.4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>21.0</td>
<td>43</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It was necessary to use a multiple comparison test (LSD method) to single out the group means that were different. Table E.20 shows the results of the test.

The group means for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier” are significantly unequal for the respondents who “Infrequently (<10 %)” used models to describe software/system architecture during their work and those who “Regularly (>50 % and <80 %)” used them in their work, Mean difference = 0.9, p-value < .05.

The group means are also unequal for the respondents who “Reasonably frequent (>15 % and <50 %)” used models to describe software/system architecture during their work compared to those who “Regularly (>50 % and <80 %)” used models in their work, Mean difference = 1.0, p-value < .05.

**Multiple Comparisons**

*Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier: LSD*

Table E.20: Multiple comparison test results using LSD method.
### APPENDIX E. COMPLEMENT INFORMATION FOR SA SURVEY

<table>
<thead>
<tr>
<th>Reasonably frequent (&gt;15% and &lt;50%)</th>
<th>Never</th>
<th>Infrequently (&lt;10%)</th>
<th>Regularly (&gt;50% and &lt;80%)</th>
<th>Nearly always (&gt;90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularly (&gt;50% and &lt;80%)</td>
<td>1.0*</td>
<td>.3</td>
<td>.01</td>
<td>.2</td>
</tr>
<tr>
<td>Nearly always (&gt;90%)</td>
<td>1.0</td>
<td>.3</td>
<td>.01</td>
<td>.2</td>
</tr>
<tr>
<td>Reasonably frequent (&gt;15% and &lt;50%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Never</td>
<td>.6</td>
<td>.3</td>
<td>.06</td>
<td>.0</td>
</tr>
<tr>
<td>Infrequently (&lt;10%)</td>
<td>.1</td>
<td>.2</td>
<td>.62</td>
<td>-.4</td>
</tr>
<tr>
<td>Regularly (&gt;50% and &lt;80%)</td>
<td>1.0*</td>
<td>.3</td>
<td>.01</td>
<td>.3</td>
</tr>
<tr>
<td>Nearly always (&gt;90%)</td>
<td>.5</td>
<td>.4</td>
<td>.15</td>
<td>-.2</td>
</tr>
</tbody>
</table>

*The mean difference is significant at the 0.05 level.

#### E.3.1.9 Analyses of Q6 with (Q9, Q10, and Q15 to Q23):

This section summarised many analysis tables results for question six into one table, as illustrated in Table E.21.
Table E.21: Independent sample t-test results for equality of two population group means of Likert scaled items Q9, Q10 and Q15 to Q23 by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture (Q6). Groups formed by the developers who agreed with a Q6 item (Q6CB1 to Q6CB8) and developers who didn’t agree with that item.

<table>
<thead>
<tr>
<th></th>
<th>Q9</th>
<th>Q10</th>
<th>Q15</th>
<th>Q16</th>
<th>Q17</th>
<th>Q18</th>
<th>Q19</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q6</td>
<td>-0.41</td>
<td>-0.36</td>
<td>-1.02</td>
<td>-0.76</td>
<td>-0.82</td>
<td>1.34</td>
<td>-0.14</td>
<td>0.24</td>
<td>-0.83</td>
<td>-0.58</td>
<td>-0.43</td>
</tr>
<tr>
<td>CB1</td>
<td>(.681)</td>
<td>(.718)</td>
<td>(.314)</td>
<td>(.454)</td>
<td>(.415)</td>
<td>(.197)</td>
<td>(.888)</td>
<td>(.814)</td>
<td>(.413)</td>
<td>(.501)</td>
<td>(.671)</td>
</tr>
<tr>
<td>Q6</td>
<td>1.17</td>
<td>0.46</td>
<td>0.11</td>
<td>-0.26</td>
<td>0.66</td>
<td>-0.45</td>
<td>-1.45</td>
<td>-0.71</td>
<td>-0.22</td>
<td>0.56</td>
<td>0.39</td>
</tr>
<tr>
<td>CB2</td>
<td>(.247)</td>
<td>(.650)</td>
<td>(.912)</td>
<td>(.794)</td>
<td>(.513)</td>
<td>(.655)</td>
<td>(.153)</td>
<td>(.483)</td>
<td>(.825)</td>
<td>(.582)</td>
<td>(.700)</td>
</tr>
<tr>
<td>Q6</td>
<td>-0.27</td>
<td>1.96</td>
<td>0.07</td>
<td>1.56</td>
<td>0.55</td>
<td>-1.13</td>
<td>2.20</td>
<td>-1.15</td>
<td>0.58</td>
<td>-0.46</td>
<td>-0.48</td>
</tr>
<tr>
<td>CB3</td>
<td>(.789)</td>
<td>(.956)</td>
<td>(.947)</td>
<td>(.126)</td>
<td>(.589)</td>
<td>(.267)</td>
<td>(.034)</td>
<td>(.257)</td>
<td>(.568)</td>
<td>(.649)</td>
<td>(.637)</td>
</tr>
<tr>
<td>Q6</td>
<td>0.88</td>
<td>-0.42</td>
<td>1.98</td>
<td>1.23</td>
<td>0.49</td>
<td>0.80</td>
<td>-0.57</td>
<td>0.86</td>
<td>-0.60</td>
<td>0.26</td>
<td>-0.38</td>
</tr>
<tr>
<td>CB4</td>
<td>(.382)</td>
<td>(.677)</td>
<td>(.054)</td>
<td>(.226)</td>
<td>(.626)</td>
<td>(.428)</td>
<td>(.569)</td>
<td>(.395)</td>
<td>(.554)</td>
<td>(.793)</td>
<td>(.706)</td>
</tr>
<tr>
<td>Q6</td>
<td>2.27</td>
<td>-1.25</td>
<td>1.00</td>
<td>-0.18</td>
<td>-0.17</td>
<td>0.79</td>
<td>1.64</td>
<td>-0.85</td>
<td>0.18</td>
<td>0.75</td>
<td>2.02</td>
</tr>
<tr>
<td>CB5</td>
<td>(.028)</td>
<td>(.218)</td>
<td>(.324)</td>
<td>(.861)</td>
<td>(.870)</td>
<td>(.436)</td>
<td>(.109)</td>
<td>(.399)</td>
<td>(.859)</td>
<td>(.460)</td>
<td>(.049)</td>
</tr>
<tr>
<td>Q6</td>
<td>4.78</td>
<td>0.00</td>
<td>0.26</td>
<td>0.49</td>
<td>0.07</td>
<td>1.47</td>
<td>0.49</td>
<td>0.97</td>
<td>0.12</td>
<td>2.16</td>
<td>-0.96</td>
</tr>
<tr>
<td>CB6</td>
<td>(.000)</td>
<td>(1.00)</td>
<td>(.795)</td>
<td>(.628)</td>
<td>(.044)</td>
<td>(.150)</td>
<td>(.624)</td>
<td>(.336)</td>
<td>(.902)</td>
<td>(.037)</td>
<td>(.343)</td>
</tr>
<tr>
<td>Q6</td>
<td>-0.46</td>
<td>-1.58</td>
<td>0.48</td>
<td>0.15</td>
<td>-0.46</td>
<td>0.99</td>
<td>1.68</td>
<td>2.55</td>
<td>0.23</td>
<td>1.27</td>
<td>-1.59</td>
</tr>
<tr>
<td>CB7</td>
<td>(.648)</td>
<td>(.121)</td>
<td>(.636)</td>
<td>(.883)</td>
<td>(.646)</td>
<td>(.326)</td>
<td>(.100)</td>
<td>(.015)</td>
<td>(.823)</td>
<td>(.212)</td>
<td>(.120)</td>
</tr>
<tr>
<td>Q6</td>
<td>-2.29</td>
<td>-0.46</td>
<td>-2.28</td>
<td>-1.62</td>
<td>0.66</td>
<td>-1.27</td>
<td>-1.58</td>
<td>-0.15</td>
<td>-0.72</td>
<td>0.00</td>
<td>-0.45</td>
</tr>
<tr>
<td>CB8</td>
<td>(.031)</td>
<td>(.650)</td>
<td>(.028)</td>
<td>(.112)</td>
<td>(.513)</td>
<td>(.212)</td>
<td>(.121)</td>
<td>(.883)</td>
<td>(.478)</td>
<td>(1.00)</td>
<td>(.658)</td>
</tr>
</tbody>
</table>

Key: t-statistic ; (p-value)
* t-statistic is significant at 5% level of significance
** t-statistic is significant at 1% level of significance

E.3.1.10 Analyses of Q7 with (Q9, Q10, and Q15 to Q23):

This section summarised many analysis tables results for question seven into one table, as illustrated in Table E.22.
Table E.22: Independent sample t-test results for equality of two population group means of Likert scaled items Q9, Q10 and Q15 to Q23 by the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture (Q7). Groups formed by the developers who agreed with a Q7 item (Q7CB1 to Q7CB5) and developers who didn’t agree with that item.

<table>
<thead>
<tr>
<th></th>
<th>Q9</th>
<th>Q10</th>
<th>Q15</th>
<th>Q16</th>
<th>Q17</th>
<th>Q18</th>
<th>Q19</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q7</td>
<td>-0.62</td>
<td>-1.82</td>
<td>0.34</td>
<td>-0.10</td>
<td>0.81</td>
<td>1.39</td>
<td>-0.16</td>
<td>1.26</td>
<td>-2.52*</td>
<td>0.44</td>
<td>-0.63</td>
</tr>
<tr>
<td>CB1</td>
<td>(.538)</td>
<td>(.075)</td>
<td>(.739)</td>
<td>(.920)</td>
<td>(.422)</td>
<td>(.171)</td>
<td>(.872)</td>
<td>(.216)</td>
<td>(.016)</td>
<td>(.666)</td>
<td>(.533)</td>
</tr>
<tr>
<td>Q7</td>
<td>0.03</td>
<td>-0.72</td>
<td>-0.55</td>
<td>1.73</td>
<td>2.34*</td>
<td>1.19</td>
<td>0.67</td>
<td>0.21</td>
<td>-1.16</td>
<td>-0.44</td>
<td>-0.43</td>
</tr>
<tr>
<td>CB2</td>
<td>(.979)</td>
<td>(.475)</td>
<td>(.586)</td>
<td>(.090)</td>
<td>(.024)</td>
<td>(.242)</td>
<td>(.508)</td>
<td>(.837)</td>
<td>(.253)</td>
<td>(.659)</td>
<td>(.671)</td>
</tr>
<tr>
<td>Q7</td>
<td>-0.37</td>
<td>0.92</td>
<td>0.57</td>
<td>0.39</td>
<td>1.12</td>
<td>-0.28</td>
<td>1.08</td>
<td>0.30</td>
<td>0.76</td>
<td>0.56</td>
<td>-1.76</td>
</tr>
<tr>
<td>CB3</td>
<td>(.710)</td>
<td>(.362)</td>
<td>(.575)</td>
<td>(.695)</td>
<td>(.268)</td>
<td>(.780)</td>
<td>(.287)</td>
<td>(.802)</td>
<td>(.451)</td>
<td>(.582)</td>
<td>(.087)</td>
</tr>
<tr>
<td>Q7</td>
<td>0.16</td>
<td>0.00</td>
<td>0.23</td>
<td>-0.61</td>
<td>0.34</td>
<td>-0.44</td>
<td>-0.57</td>
<td>-0.17</td>
<td>1.23</td>
<td>0.51</td>
<td>1.23</td>
</tr>
<tr>
<td>CB4</td>
<td>(.877)</td>
<td>(1.00)</td>
<td>(.818)</td>
<td>(.548)</td>
<td>(.735)</td>
<td>(.666)</td>
<td>(.569)</td>
<td>(.866)</td>
<td>(.226)</td>
<td>(.612)</td>
<td>(.224)</td>
</tr>
<tr>
<td>Q7</td>
<td>-0.03</td>
<td>1.17</td>
<td>-1.56</td>
<td>-1.96</td>
<td>1.94</td>
<td>-0.85</td>
<td>-1.61</td>
<td>0.50</td>
<td>1.00</td>
<td>1.43</td>
<td>0.87</td>
</tr>
<tr>
<td>CB5</td>
<td>(.977)</td>
<td>(.248)</td>
<td>(.126)</td>
<td>(.056)</td>
<td>(.060)</td>
<td>(.398)</td>
<td>(.115)</td>
<td>(.619)</td>
<td>(.324)</td>
<td>(.163)</td>
<td>(.388)</td>
</tr>
</tbody>
</table>

Key: t-statistic ; (p-value)
* t-statistic is significant at 5% level of significance

E.3.1.11 Analyses of (Q10 and Q17):

A Chi-square test was used to test the independence between the categorical variables “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “Most of the existing software architecture evaluation methods, produce qualitative results”.

Table E.23: Pearson Chi-square test results for analyses of questions Q10 and Q17.

<table>
<thead>
<tr>
<th>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</th>
<th>Most of the existing software architecture evaluation methods, produce qualitative results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>30.52</td>
</tr>
<tr>
<td>df</td>
<td>16</td>
</tr>
<tr>
<td>Sig.</td>
<td>.015</td>
</tr>
</tbody>
</table>

According to the results presented in Table E.24 and E.23, the test is significant, \( \chi^2(16) = 30.52, \) p-value < .05. This confirms that there is an association or dependence between the categorical variables “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “Most of the existing software architecture evaluation methods, produce qualitative results”.

354
Table E.24: Cross tabulation of “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “Most of the existing software architecture evaluation methods, produce qualitative results”.

<table>
<thead>
<tr>
<th>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</th>
<th>Most of the existing software architecture evaluation methods, produce qualitative results.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>Strongly Disagree</td>
<td>0</td>
</tr>
<tr>
<td>Disagree</td>
<td>1</td>
</tr>
<tr>
<td>Neutral</td>
<td>0</td>
</tr>
<tr>
<td>Agree</td>
<td>0</td>
</tr>
<tr>
<td>Strongly Agree</td>
<td>0</td>
</tr>
</tbody>
</table>

E.3.1.12 Analyses of (Q13 and Q23):

Table E.25: Cross tabulation of the most important factors that could SUPPORT quantitative evaluation for any SA and the opinion about the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”.

<table>
<thead>
<tr>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
<th>Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>The language used for describing SA</td>
<td>0</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>3</td>
</tr>
<tr>
<td>Using standard language and architecture framework for describing SA</td>
<td>0</td>
</tr>
<tr>
<td>Tools availability for describing and evaluating SA</td>
<td>2</td>
</tr>
<tr>
<td>Documenting mechanism used during SA description</td>
<td>0</td>
</tr>
</tbody>
</table>

A Chi-square test was used to test the independence between two considered categorical variables. The test is significant at 6% level, $\chi^2(20) = 30.75$, p-value < .06. Therefore, there is an association or
dependence between the most important factors that could SUPPORT quantitative evaluation for any SA and the opinion about the statement “Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes”.

Table E.26: Pearson Chi-square test results for analyses of questions Q13 and Q22.

<table>
<thead>
<tr>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
<th>Current technology allows us to develop general software evaluation models that assess any software architecture against any quality attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>30.75</td>
</tr>
<tr>
<td>df</td>
<td>20</td>
</tr>
<tr>
<td>Sig.</td>
<td>.059</td>
</tr>
</tbody>
</table>

E.3.1.13 Analyses of (Q10 and Q13):

A Chi-square test was used to test the independence between two categorical variables: “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “the most important factors that could SUPPORT quantitative evaluation for any SA”.

Table E.27: Pearson Chi-square test results for analyses of questions Q10 and Q13.

<table>
<thead>
<tr>
<th>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</th>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>34.36</td>
</tr>
<tr>
<td>df</td>
<td>20</td>
</tr>
<tr>
<td>Sig.</td>
<td>.024</td>
</tr>
</tbody>
</table>

The test yielded significant results, $\chi^2(20) = 34.36$, p-value < .05. This led to the conclusion that there is a significant association or dependence between the categorical variables “Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation” and “the most important factors that could SUPPORT quantitative evaluation for any SA”.

E.3.2 Examples of non-significant results for 2-dimensions analyses

E.3.2.1 Analyses of (Q2 and Q6)

There is no association or dependence between sectors in which respondents gained most of their general software development experience and the main factors that encourage the utilisation of modelling techniques to describe software/system architecture, $\chi^2 = 13.94$, p-value > .05.

Table E.28: Cross tabulation of respondent's sectors in which respondents gained most of their general software development experience and the main factors that encourage the utilisation of modelling techniques to describe software/system architecture.

| In which of the following sectors have you gained most of your general software development experience? | What are the main factors that ENCOURAGE the utilisation of modelling techniques to describe software/system architecture? You may select up to two of the following options. |
### E.3. TWO DIMENSIONS MATRICES ANALYSIS

<table>
<thead>
<tr>
<th></th>
<th>Academia</th>
<th>Industry</th>
<th>Government</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
<td>Count 14</td>
<td>Count 11</td>
<td>Count 5</td>
</tr>
<tr>
<td>Most available architecture modelling references are clear and</td>
<td>Count 2</td>
<td>Count 3</td>
<td>Count 3</td>
</tr>
<tr>
<td>well documented, which helps developers understand and apply the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modelling approach easily</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It makes the designers/programers job much easier</td>
<td>Count 10</td>
<td>Count 3</td>
<td>Count 1</td>
</tr>
<tr>
<td>Makes the evaluation of stakeholders requirements for quality</td>
<td>Count 6</td>
<td>Count 4</td>
<td>Count 1</td>
</tr>
<tr>
<td>attributes possible in the early stages of the development</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lifecycle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliable modelling tools for describing the architecture exist,</td>
<td>Count 3</td>
<td>Count 1</td>
<td>Count 0</td>
</tr>
<tr>
<td>which makes the usability factor much easier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The wide range of modelling language formalities (from informal</td>
<td>Count 1</td>
<td>Count 1</td>
<td>Count 0</td>
</tr>
<tr>
<td>models to formal), makes the selection of architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>description technique more feasible</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural models can be compiled to produce a real</td>
<td>Count 2</td>
<td>Count 3</td>
<td>Count 1</td>
</tr>
<tr>
<td>functioning software/system with existing modelling languages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and tools, (e.g. SysML, UML)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teaching of the architecture modelling languages in the</td>
<td>Count 2</td>
<td>Count 4</td>
<td>Count 2</td>
</tr>
<tr>
<td>academic sectors</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Academia</th>
<th>Industry</th>
<th>Government</th>
</tr>
</thead>
<tbody>
<tr>
<td>In which of the following sectors have you gained most of your</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general software development experience?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>What are the main factors that ENCOURAGE the utilisation of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modelling techniques to describe software/system architecture?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>You may select up to two of the following options.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chi-square</td>
<td>13.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>df</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sig.</td>
<td>.604</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table E.29: Pearson Chi-square test results for analyses of questions Q2 and Q6.
E.3.2.2 Analyses of (Q2 and Q7):

There is no association or dependence between sectors in which respondents gained most of their general software development experience and the main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture, $\chi^2 = 11.64$, p-value > .05.

Table E.30: Cross tabulation of respondent’s sectors in which respondents gained most of their general software development experience and the main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture.

| What are the main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture? You may select up to two options. | In which of the following sectors have you gained most of your general software development experience? |
|---|---|---|
| Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system | Count | 10 | 6 | 3 |
| Lack of standardisation between existing architecture modelling technique, notations, and semantics | Count | 10 | 6 | 1 |
| Current architecture description languages (including modelling languages) are still immature | Count | 4 | 3 | 1 |
| Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money | Count | 4 | 5 | 3 |
| Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance) | Count | 6 | 2 | 5 |
### E.3. TWO DIMENSIONS MATRICES ANALYSIS

Table E.31: Pearson Chi-square test results for analyses of questions Q2 and Q7.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture? You may select up to two options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>11.46</td>
</tr>
<tr>
<td>df</td>
<td>10</td>
</tr>
<tr>
<td>Sig.</td>
<td>.323</td>
</tr>
</tbody>
</table>

**E.3.2.3 Analyses of (Q2 and Q13):**

There is no significant association between these two items, $\chi^2 = 7.68$, p-value > .05.

Table E.32: Pearson Chi-square test results for analyses of questions Q2 and Q13.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>7.68</td>
</tr>
<tr>
<td>df</td>
<td>10</td>
</tr>
<tr>
<td>Sig.</td>
<td>.660</td>
</tr>
</tbody>
</table>

Table E.33: Cross tabulation of sectors in which respondents gained most of their general software development experience and the most important factors that could SUPPORT quantitative evaluation for any SA.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>What are the most important factors that could SUPPORT quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>Formality level of SA description</td>
</tr>
<tr>
<td></td>
<td>Using standard language and architecture framework for describing SA</td>
</tr>
<tr>
<td></td>
<td>Tools availability for describing and evaluating SA</td>
</tr>
<tr>
<td></td>
<td>Documenting mechanism used during SA description</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Academia</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Industry</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Government</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

**E.3.2.4 Analyses of (Q2 and Q14):**

There is no significant association between these two items, $\chi^2 = 8.93$, p-value > .05. Thus the sectors in which respondents gained most of their software development experience does not seem to shape the respondents’ opinions about the factors that hinder quantitative evaluation of SA.
Table E.34: Cross tabulation of sectors in which respondents gained most of their general software development experience and the most important factors that could HINDER quantitative evaluation for any Software Architecture (SA).

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>What are the most important factors that could HINDER quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The language used for describing SA</td>
<td>The language used for describing SA</td>
</tr>
<tr>
<td>Formality level of SA description</td>
<td>Using standard language and architecture framework for describing SA</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Academia</td>
<td>10</td>
</tr>
<tr>
<td>Industry</td>
<td>4</td>
</tr>
<tr>
<td>Government</td>
<td>3</td>
</tr>
</tbody>
</table>

Table E.35: Pearson Chi-square test results for analyses of questions Q2 and Q14.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>What are the most important factors that could HINDER quantitative evaluation for any SA? You may choose two.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>8.93</td>
</tr>
<tr>
<td>df</td>
<td>10</td>
</tr>
<tr>
<td>Sig.</td>
<td>.539</td>
</tr>
</tbody>
</table>

E.4 Examples for three and four dimensional analyses

Within this section, examples of (3 and 4) dimensional analysis are illustrated. However, many tests and analysis (which not included here) for both dimensions were performed, but their results are inappropriate due to the same reasons that have been reported within these examples.

E.4.1 Three dimensional analysis:

1- Analyses of Q3 by Q5 when Q1: What is your general field of expertise regarding software development? = Requirements elicitation / modelling / analysis The test is significant at 5% level of significance, \( \chi^2 = 26.62, p-value < .05 \). But the test result may invalid due to violation of the assumption of minimum expected cell counts for a valid non-parametric Chi-Square test. Though the test result is significant at 5% level that signifies the association between years of experiences of respondents in the software/systems development field and how often the respondents use models to describe software/system architecture during their work when the general field of expertise regarding software development is “Requirements elicitation/modelling/analysis” but due to low expected cell counts, the result is invalid and so on for all the three dimensional crosstabulations.

2- Analyses of Q22 by Q2 when Q1: What is your general field of expertise regarding software development? = Project management. One way ANOVA procedure is used to test the difference among group means for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier” corresponding to the different sectors from where the respondents gained their general software development experience and for the respondents whose general field of expertise regarding software development is project management.
Table E.37: Cross tabulation of years of experiences of respondents in the software/systems development field and how often the respondents use models to describe software/system architecture during their work when the general field of expertise regarding software development is “Requirements elicitation/modelling/analysis”.

<table>
<thead>
<tr>
<th>How many years’ experience do you have in total in the software/systems development field?</th>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>Count</th>
<th>Expected Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Never (≤ 10%)</td>
<td>Infrequently (15% and &lt; 50%)</td>
<td>Reasonably frequently (50% and &lt; 80%)</td>
</tr>
<tr>
<td>5–10 (years)</td>
<td>Count</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>1.1</td>
<td>4.5</td>
</tr>
<tr>
<td>10–15 (years)</td>
<td>Count</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>.3</td>
<td>1.4</td>
</tr>
<tr>
<td>15–20 (years)</td>
<td>Count</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>.1</td>
<td>.3</td>
</tr>
<tr>
<td>20–25 (years)</td>
<td>Count</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>Count</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>.2</td>
<td>.7</td>
</tr>
<tr>
<td>Total</td>
<td>Count</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Expected Count</td>
<td>2.0</td>
<td>8.0</td>
</tr>
</tbody>
</table>

 Chi-square test results for testing association between years of experiences of respondents in the software/systems development field and how often the respondents use models to describe software/system architecture during their work when the general field of expertise regarding software development is “Requirements elicitation/modelling/analysis”.

- Chi-square statistic is significant at the .05 level.
- More than 20% of cells in this subtable have expected cell counts less than 5. Chi-square results may be invalid.
- The minimum expected cell count in this subtable is less than 1. Chi-square results may be invalid.

Results are based on nonempty rows and columns in each innermost subtable.
Table E.38: Multiple Comparisons between Q2, Q22, when Q1= Project Management

Dependent Variable: Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier.

LSD

<table>
<thead>
<tr>
<th></th>
<th>Mean Difference (I-J)</th>
<th>Std. Error</th>
<th>Sig.</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I) In which of the following sectors have you gained most of your general software development experience?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Academia</td>
<td>.500</td>
<td>.274</td>
<td>.015</td>
<td>.30</td>
<td>1.70</td>
</tr>
<tr>
<td>Industry</td>
<td>.500</td>
<td>.274</td>
<td>.015</td>
<td>-1.70</td>
<td>-0.30</td>
</tr>
<tr>
<td>Government</td>
<td>.500</td>
<td>.274</td>
<td>.015</td>
<td>-1.20</td>
<td>.20</td>
</tr>
</tbody>
</table>

* The mean difference is significant at the 0.05 level.

The test is significant for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier”, F = 6.9, p-value < .05. So, the group means corresponding to the different sectors where the respondents gained their general software development experience and for the respondents whose general field of expertise regarding software development is project management are statistically different for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier”.

Now to see which group means are statistically different, a multiple comparison test (LSD method) is used.

The group means for the item “Restricting the description of architecture to a specific modelling language during development, should make the architecture quantitative evaluation easier” is significantly different for the respondents whose general field of expertise regarding software development is project management and who gained there general software development experience in “Academia” than who gained there general software development experience in “Industry”, mean = 1.0, p-value <.05.

Table E.39: One way ANOVA table to test the difference among group means for Q22, Q2, when Q1= Project Management

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>1.375</td>
<td>2</td>
<td>.688</td>
<td>6.875</td>
<td>.037</td>
</tr>
<tr>
<td>Within Groups</td>
<td>.500</td>
<td>5</td>
<td>.100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1.875</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. Project management = Yes
E.4.2 Example for Four dimensional analyses

Due to low frequency in four dimensional analyses, all the results are either invalid or non-significant. The following table shows the cross tabulation of Q2 and Q3 with Q4 and Q5 where most of the cell counts are either 0 or 1. Therefore, four dimensional analyses for this data will not be valid.

Table E.40: Cross tabulation Analyses for (Q2 and Q3 with Q4 and Q5).

<table>
<thead>
<tr>
<th>How many years experience do you have in total in the software/systems development field?</th>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>No</th>
<th>Yes, which language?</th>
<th>Never</th>
<th>Infrequently (&lt; 10%)</th>
<th>Reasonably frequently (&gt; 15% and &lt; 50%)</th>
<th>Regularly (&gt; 50% and &lt; 80%)</th>
<th>Nearly always (&gt; 90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5–10 (years)</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
<td>Academia</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Industry</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10–15 (years)</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
<td>Academia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Industry</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>15–20 (years)</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
<td>Academia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20–25 (years)</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
<td>Academia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Over 25 (years)</td>
<td>In which of the following sectors have you gained most of your general software development experience?</td>
<td>Academia</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Industry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Government</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
E.5  Supported information for the Summary Table of Chapter 5

This section includes all tables mentioned by the summary Table 5.13 in Chapter 5, which have not mentioned by other sections.

E.5.1 Analyses of (Q2 and Q11):

There is a significant association or dependence between sectors in which respondents gained most of their general software development experience with whether the developers were aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models, \(\chi^2 = 5.43, \text{ p-value < .07,}\) leading us to believe that some sectors have applied more effort than others in creating awareness about SA evaluation.

Table E.41: Cross tabulation of sectors in which respondents gained most of their general software development experience and whether respondents were aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models (e.g. detecting attacks for security).

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models, (e.g. detecting attacks for security).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Count</td>
<td>Count</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Academia</td>
<td>22</td>
</tr>
<tr>
<td>Industry</td>
<td>11</td>
</tr>
<tr>
<td>Government</td>
<td>7</td>
</tr>
</tbody>
</table>

Table E.42: Pearson Chi-square test results for analyses of questions Q2 and Q11.

<table>
<thead>
<tr>
<th>In which of the following sectors have you gained most of your general software development experience?</th>
<th>Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models, (e.g. detecting attacks for security).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>5.43</td>
</tr>
<tr>
<td>df</td>
<td>2</td>
</tr>
<tr>
<td>Sig.</td>
<td>.066</td>
</tr>
</tbody>
</table>

E.5.2 Analyses of (Q3 and Q5):

The results in Table E.44 and Table E.43 reveal a significant association between the years of experience in total in the software/systems development field with how they used models to describe software/system architecture during their work, \(\chi^2 = 26.18, \text{ p-value > .05,}\)
E.5. SUPPORTED INFORMATION FOR THE SUMMARY TABLE OF …

Table E.43: Pearson Chi-square test results for analyses of questions Q3 and Q5.

<table>
<thead>
<tr>
<th>How many years’ experience do you have in total in the software/systems development field?</th>
<th>How often do you use models to describe software/system architecture during your work?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>26.18</td>
</tr>
<tr>
<td>df</td>
<td>16</td>
</tr>
<tr>
<td>Sig.</td>
<td>.052</td>
</tr>
</tbody>
</table>

Table E.44: Cross tabulation of respondent’s years of experience in total in the software/systems development field and how often they used models to describe software/system architecture during your work.

| How many years’ experience do you have in total in the software/systems development field? | How often do you use models to describe software/system architecture during your work? |
|---|---|---|---|---|---|
| Never | Infrequently (< 10%) | Reasonably frequently (> 15% and < 50%) | Regularly (> 50% and < 80%) | Nearly always (> 90%) |
| Count | Count | Count | Count | Count |
| 5–10 (years) | 3 | 10 | 8 | 2 | 1 |
| 10–15 (years) | 3 | 1 | 6 | 1 | 0 |
| 15–20 (years) | 0 | 2 | 1 | 0 | 1 |
| 20–25 (years) | 0 | 4 | 1 | 2 | 0 |
| Over 25 (years) | 1 | 1 | 0 | 0 | 2 |

E.5.3 Analyses of (Q3 and Q7):

There is a significant association between years of experience in total in the software/systems development field and the main factors that DISCOURAGE the utilisation of modelling techniques to describe software/system architecture, \( \chi^2 = 30.44, p\text{-value} < .07 \). The full analysis of the results is shown in Table 5.7 and E.45.

Table E.45: Pearson Chi-square test results for analyses of questions Q3 and Q7.

<table>
<thead>
<tr>
<th>How many years’ experience do you have in total in the software/systems development field?</th>
<th>What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture? You may select up to two options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>30.44</td>
</tr>
<tr>
<td>df</td>
<td>20</td>
</tr>
<tr>
<td>Sig.</td>
<td>.063</td>
</tr>
</tbody>
</table>
E.5.4 Analyses of (Q9 and Q10 by Q6)

Table E.46: Independent samples t-test results for the selected items for equality of grouped population means.

<table>
<thead>
<tr>
<th>Item: It makes the designers/programers job much easier. Groups: “Yes” and “No”</th>
<th>Levene’s Test for Equality of Variances</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>Sig.</td>
<td>t</td>
</tr>
<tr>
<td>Developing software/system architecture using current architectural frameworks (e.g. ISO/IEC 42010, DoDAF, RUP/4+1) increases the reliability, standardisation, and reusability of the resulting architecture.</td>
<td>4.1</td>
<td>.05</td>
<td>-.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Equal variances assumed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Equal variances not assumed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-3</td>
<td>39.2</td>
</tr>
<tr>
<td>Usage of software style/pattern concepts &amp; models during architecture development, increases the utilisation of modelling description languages, BUT decreases the simplicity of the architecture valuation.</td>
<td>1.7</td>
<td>.20</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Equal variances assumed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Equal variances not assumed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.8</td>
<td>21.1</td>
</tr>
</tbody>
</table>

The independent sample t-test result is non-significant for the item “Developing software/system architecture using current architectural frameworks ...etc.”, \( t(43) = -0.2, \ p-value > .05 \).

The test is also non-significant for the item “Usage of software style/pattern concepts & models during architecture ...etc.”, \( t(44) = 2.0, \ p-value > .05 \). This leads to the conclusion that there was no significant difference between respondents who answered “Yes” and those who answered “No” to the item “It makes the designers/programmers job much easier”.

Table E.46: Independent samples t-test results for the selected items for equality of grouped population means.
### E.5.5 Additional important summary tables for 2-Dimensional analyses

In this section I have summarised and combined several important results, from many different tables, to allow the readers to be able to view the overall analyses picture. However, most of the summary tables included in this section are long and stuffed with valuable information (which could be a disadvantage); but they should be better for representing these results, rather-than reporting them via (70 pages) as a separated tables.

Table E.47: Cross tabulation of the developer’s years’ of experience in the software/system development field against selected categorical questions and corresponding results for $\chi^2$-test of independence.

<table>
<thead>
<tr>
<th>Are you aware of any software/system architectural description/modelling languages?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many years’ experience do you have in total in the software/systems development field?</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>5–10 (years)</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>10–15 (years)</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>15–20 (years)</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20–25 (years)</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>&gt;25 (years)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$\chi^2$ (p-value)

5.26 (.261)

<table>
<thead>
<tr>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>Never</th>
<th>Infrequently (&lt;10%)</th>
<th>Reasonably frequent (&gt;15% and &lt;50%)</th>
<th>Regularly (&gt;50% and &lt;80%)</th>
<th>Nearly always (&gt;90%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

$\chi^2$ (p-value)

26.18 (.052)

<table>
<thead>
<tr>
<th>What are the main factors that encourage the utilization of modelling techniques to describe software/system architecture?</th>
<th>Easier to demonstrate the software/system concept and features</th>
<th>Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily</th>
<th>It makes the designers/programmers job much easier</th>
<th>Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development life cycle</th>
<th>Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$\chi^2$ (p-value)

36.62 (.263)
The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible

| | 1 | 1 | 0 | 0 | 0 |

Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, X-UML)

| | 2 | 1 | 0 | 2 | 1 |

Teaching of the architecture modelling languages in the academic sectors

| | 3 | 3 | 0 | 1 | 1 |

What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture?

| | 8 | 3 | 3 | 4 | 1 |

Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system

| | 7 | 1 | 2 | 4 | 3 |

Lack of standardisation between existing architecture modelling technique, notations, and semantics

| | 0 | 4 | 1 | 2 | 1 |

Current architecture description languages (including modelling languages) are still immature

| | 7 | 4 | 0 | 1 | 0 |

Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money

| | 8 | 4 | 0 | 0 | 1 |

Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)

| | 3 | 1 | 0 | 0 | 0 |

Natural language (e.g. English-text) only

| | 5 | 2 | 0 | 0 | 1 | 29.83 (.191) |

Semi-formal language (e.g. UML, SysML) only

| | 0 | 0 | 0 | 0 | 1 |

Formal language (e.g. ADLs, Z) only

| | 1 | 0 | 0 | 1 | 0 |

Formal language & Natural language together

<p>| | 5 | 8 | 4 | 5 | 2 |</p>
<table>
<thead>
<tr>
<th>Semi-formal language &amp; Formal language together</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>All three languages above together</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models</td>
<td>Yes</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>20</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?</td>
<td>Yes</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>21</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>What are the most important factors that could SUPPORT quantitative evaluation for any SA?</td>
<td>The language used for describing SA</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Formality level of SA description</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Using standard language and architecture framework for describing SA</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Tools availability for describing and evaluating SA</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Documenting mechanism used during SA description</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>What are the most important factors that could HINDER quantitative evaluation for any SA</td>
<td>The language used for describing SA</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Formality level of SA description</td>
<td>10</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Using standard language and architecture framework for describing SA</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Tools availability for describing and evaluating SA</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Documenting mechanism used during SA description</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table E.48: Cross tabulation between the developer’s awareness regarding modelling languages and selected categorical questions.
### How often do you use models to describe software/system architecture during your work?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Count</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Infrequently (&lt;10%)</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Reasonably frequent (&gt;15% and &lt;50%)</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Regularly (&gt;50% and &lt;80%)</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Nearly always (&gt;90%)</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**Chi-square (df=4, p=0.777):** 1.77

### What are the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture?

<table>
<thead>
<tr>
<th>Factor</th>
<th>Count</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>It makes the designers/programmers job much easier</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development lifecycle</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, X UML)</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Teaching of the architecture modelling languages in the academic sectors</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

**Chi-square (df=7, p=0.755):** 5.03

### What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture?

<table>
<thead>
<tr>
<th>Factor</th>
<th>Count</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Lack of standardisation between existing architecture modelling technique, notations, and semantics</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

**Chi-square (df=3, p=0.293):** 6.14
E.5. SUPPORTED INFORMATION FOR THE SUMMARY TABLE OF …

<table>
<thead>
<tr>
<th>Current architecture description languages (including modelling languages) are still immature</th>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

From your experience, what is the best language to use to describe software/system architecture, so as to be more useful to all stakeholders, and to be easier to undertake qualitative and quantitative assessments?

| Natural language (e.g. English-text) only | 2 | 2 |
| Semi-formal language (e.g. UML, SysML) only | 8 | 0 |
| Formal language (e.g. ADLs, Z) only | 0 | 1 |
| Formal language & Natural language together | 2 | 0 |
| Semi-formal language & Natural language together | 19 | 4 |
| Semi-formal language & Formal language together | 1 | 0 |
| All three languages above together | 6 | 0 |

Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models?

| Yes | 7 | 0 |
| No | 31 | 8 |

Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?

| Yes | 5 | 0 |
| No | 32 | 8 |

What are the most important factors that could SUPPORT quantitative evaluation for any SA?

| The language used for describing SA | 6 | 0 |
| Formality level of SA description | 10 | 2 |
| Using standard language and architecture framework for describing SA | 15 | 5 |
| Tools availability for describing and evaluating SA | 18 | 4 |
| Documenting mechanism used during SA description | 5 | 3 |

What are the most important factors that could HINDER quantitative evaluation for any SA?

| The language used for describing SA | 13 | 4 |
| Formality level of SA description | 16 | 4 |
| Using standard language and architecture framework for describing SA | 4 | 1 |
| Tools availability for describing and evaluating SA | 13 | 2 |
| Documenting mechanism used during SA description | 6 | 0 |
Table E.49: Cross tabulation of how often developer’s used models to describe software/system architecture during their work against selected categorical questions and corresponding results for $\chi^2$ test of independence.

<table>
<thead>
<tr>
<th>How often do you use models to describe software/system architecture during your work?</th>
<th>Never</th>
<th>Infrequently (&lt;10%)</th>
<th>Reasonably frequent (15–50%)</th>
<th>Regularly (50–80%)</th>
<th>Nearly always (&gt;90%)</th>
<th>$\chi^2$ (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>56.46 (.005)</td>
</tr>
</tbody>
</table>

What are the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture?

<table>
<thead>
<tr>
<th>Factor</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
<td>5</td>
</tr>
<tr>
<td>Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily</td>
<td>0</td>
</tr>
<tr>
<td>It makes the designers/programers job much easier</td>
<td>0</td>
</tr>
<tr>
<td>Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development life cycle</td>
<td>0</td>
</tr>
<tr>
<td>Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier</td>
<td>0</td>
</tr>
<tr>
<td>The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible</td>
<td>0</td>
</tr>
<tr>
<td>Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, xUML)</td>
<td>0</td>
</tr>
<tr>
<td>Teaching of the architecture modelling languages in the academic sectors</td>
<td>4</td>
</tr>
</tbody>
</table>

What are the main factors that DISCOURAGE the utilization of modelling techniques to describe software/system architecture?
### E.5. SUPPORTED INFORMATION FOR THE SUMMARY TABLE OF ...

<table>
<thead>
<tr>
<th>Hard to integrate these models with other artefacts (e.g. Design models), so they become standalone models, which to some degree are not that useful during the development of software/system</th>
<th>1</th>
<th>8</th>
<th>8</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of standardisation between existing architecture modelling technique, notations, and semantics</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Current architecture description languages (including modelling languages) are still immature</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Modelling the architecture has limited benefit to the whole software/system development process, so it’s to some extent a waste of time and money</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hard to evaluate architecture models against any stakeholder’s quality attributes (e.g. Security, performance)</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**From your experience, what is the best language to use to describe software/system architecture, so as to be more useful to all stakeholders, and to be easier to undertake qualitative and quantitative assessments?**

<table>
<thead>
<tr>
<th>Natural language (e.g. English-text) only</th>
<th>2</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-formal language (e.g. UML, SysML) only</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Formal language (e.g. ADLs, Z) only</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Formal language &amp; Natural language together</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Semi-formal language &amp; Natural language together</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Semi-formal language &amp; Formal language together</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>All three languages above together</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models?**

| Yes | 1 | 0 | 3 | 2 | 1 | 5.62 (.229) |
| No | 6 | 15 | 13 | 3 | 3 |

**Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?**
Table E.50: Cross tabulation of the developer’s general field of expertise regarding software development against selected categorical questions and corresponding results for $\chi^2$ test of independence.

<table>
<thead>
<tr>
<th>What is your general field of expertise regarding software development? You may select more than one of the following options.</th>
<th>( \chi^2 ) (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements elicitation/modelling/analysis</td>
<td>Project management</td>
</tr>
<tr>
<td>Requirements elicitation/modelling/analysis</td>
<td>20</td>
</tr>
</tbody>
</table>

Are you aware of any software/system architectural description/modelling languages

| Yes | 20 | 8 | 18 | 27 | 25 | 15 | 7 | 4 |
| No | 3 | 3 | 1 | 3 | 3 | 3 | 1 | 2 | 9.69 | (.287) |
In which of the following sectors have you gained most of your general software development experience?

<table>
<thead>
<tr>
<th>Sector</th>
<th>9</th>
<th>4</th>
<th>7</th>
<th>14</th>
<th>13</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>Total (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academia</td>
<td>4</td>
<td>7</td>
<td>14</td>
<td>13</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
<td>12.49 (.710)</td>
</tr>
<tr>
<td>Industry</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>11</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Government</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

From your experience, what is the best language to use to describe software/system architecture, so as to be more useful to all stakeholders, and to be easier to undertake qualitative and quantitative assessments?

<table>
<thead>
<tr>
<th>Language Type</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>Total (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural langu</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Semi-formal</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Formal only</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Formal lang</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Form &amp; Natu</td>
<td>13</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>15</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>53.21 (.281)</td>
</tr>
<tr>
<td>Semi-formal</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Sem &amp; Forma</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Are you aware of any system/software architectural tactics or metrics that have been or are being used for evaluating architecture description models?

<table>
<thead>
<tr>
<th>Yes</th>
<th>5</th>
<th>2</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>0</th>
<th>Total (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>18</td>
<td>9</td>
<td>14</td>
<td>25</td>
<td>22</td>
<td>12</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Do you know or use any architectural evaluation method that can produce quantitative measures surrounding architecture characteristics?

<table>
<thead>
<tr>
<th>Yes</th>
<th>18</th>
<th>10</th>
<th>14</th>
<th>26</th>
<th>24</th>
<th>14</th>
<th>6</th>
<th>5</th>
<th>Total (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table E.51: Independent sample t-test for equality of two population group means of "developing software/system architecture using current architectural frameworks increases the reliability, standardisation, and reusability of the resulting architecture" by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture.
### Table E.52: Independent sample t-test for equality of two population group means of "Usage of software style/pattern concepts & models during architecture development, increases the utilisation of modeling description languages, BUT decreases the simplicity of the architecture valuation" by the main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture.

<table>
<thead>
<tr>
<th>Main Factors</th>
<th>N</th>
<th>Mean</th>
<th>SD</th>
<th>t-statistic (p-value)</th>
<th>95% CI of the difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier to demonstrate the software/system concept and features</td>
<td>Yes</td>
<td>30</td>
<td>3.43</td>
<td>-0.41 (.718)</td>
<td>(-0.59, 0.39)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>15</td>
<td>3.53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily</td>
<td>Yes</td>
<td>8</td>
<td>3.75</td>
<td>1.17 (.247)</td>
<td>(-0.25, 0.94)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>37</td>
<td>3.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>It makes the designers/programers job much easier</td>
<td>Yes</td>
<td>14</td>
<td>3.43</td>
<td>-0.27 (.789)</td>
<td>(-0.47, 0.36)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>31</td>
<td>3.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development life cycle</td>
<td>Yes</td>
<td>9</td>
<td>3.67</td>
<td>0.88 (.382)</td>
<td>(-0.32, 0.82)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>36</td>
<td>3.42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier</td>
<td>Yes</td>
<td>4</td>
<td>4.25</td>
<td>2.27 (.028)</td>
<td>(0.04, 1.62)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>41</td>
<td>3.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible</td>
<td>Yes</td>
<td>2</td>
<td>4.00</td>
<td>1.02 (.313)</td>
<td>(-0.55, 1.66)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>43</td>
<td>3.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, xUML)</td>
<td>Yes</td>
<td>6</td>
<td>3.33</td>
<td>-0.46 (.648)</td>
<td>(-0.83, 0.52)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>39</td>
<td>3.49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teaching of the architecture modelling languages in the academic sectors</td>
<td>Yes</td>
<td>8</td>
<td>3.13</td>
<td>-2.29 (.031)</td>
<td>(-0.79, -0.04)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>37</td>
<td>3.54</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The main factors that ENCOURAGE the utilization of modelling techniques to describe software/system architecture.
<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3.06</td>
<td>0.68</td>
<td>0.46</td>
<td>(-0.52, 0.82)</td>
</tr>
<tr>
<td>E.5. SUPPORTED INFORMATION FOR THE SUMMARY TABLE OF ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most available architecture modelling references are clear and well documented, which helps developers understand and apply the modelling approach easily

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>38</td>
<td>3.13</td>
<td>0.64</td>
<td>1.96</td>
<td>(-0.52, 1.04)</td>
</tr>
</tbody>
</table>

It makes the designers/programers job much easier

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>32</td>
<td>3.36</td>
<td>0.93</td>
<td>1.96</td>
<td>(-0.52, 1.04)</td>
</tr>
</tbody>
</table>

Makes the evaluation of stakeholders requirements for quality attributes possible in the early stages of the development life cycle

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>36</td>
<td>2.90</td>
<td>0.88</td>
<td>-0.42</td>
<td>(-0.74, 0.49)</td>
</tr>
</tbody>
</table>

Reliable modelling tools for describing the architecture exist, which makes the usability factor much easier

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>42</td>
<td>2.50</td>
<td>1.29</td>
<td>-1.25</td>
<td>(-1.43, 0.34)</td>
</tr>
</tbody>
</table>

The wide range of modelling language formality (from informal models to formal), makes the selection of architecture description technique more feasible

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>44</td>
<td>3.00</td>
<td>0.84</td>
<td>0.00</td>
<td>(-1.24, 1.24)</td>
</tr>
</tbody>
</table>

Architectural models can be compiled to produce a real functioning software/system with existing modelling languages and tools, (e.g. SysML, xtUML)

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>36</td>
<td>2.50</td>
<td>0.55</td>
<td>-1.58</td>
<td>(-1.31, 0.16)</td>
</tr>
</tbody>
</table>

Teaching of the architecture modelling languages in the academic sectors

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
<th>Mean</th>
<th>SD</th>
<th>t-value</th>
<th>(Confidence Interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>38</td>
<td>2.88</td>
<td>0.64</td>
<td>-0.46</td>
<td>(-0.82, 0.52)</td>
</tr>
</tbody>
</table>
Table E.53: Summary table of $\chi^2$-test results for testing pairwise independence between categorical variables.

<table>
<thead>
<tr>
<th></th>
<th>Q2</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
<th>Q14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>12.49 (.710)</td>
<td>9.69 (.287)</td>
<td></td>
<td></td>
<td>53.21 (.281)</td>
<td>13.94 (.083)</td>
<td>9.40 (.309)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>0.74 (.692)</td>
<td>9.64 (.291)</td>
<td>13.94 (.604)</td>
<td>11.46 (.323)</td>
<td>6.44 (.893)</td>
<td>5.43 (.066)</td>
<td>0.23 (.892)</td>
<td>7.68 (.660)</td>
<td>8.93 (.539)</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>5.26 (.261)</td>
<td>26.18 (.052)</td>
<td>36.62 (.263)</td>
<td>30.44 (.063)</td>
<td>29.83 (.191)</td>
<td>2.95 (.566)</td>
<td>3.29 (.511)</td>
<td>12.94 (.880)</td>
<td>17.47 (.622)</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td>1.77 (.777)</td>
<td>5.03 (.755)</td>
<td>6.14 (.293)</td>
<td>12.23 (.057)</td>
<td>1.74 (.187)</td>
<td>1.22 (.270)</td>
<td>4.79 (.442)</td>
<td>2.81 (.730)</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td>56.46** (.005)</td>
<td>33.54* (.029)</td>
<td>23.44 (.494)</td>
<td>5.62 (.229)</td>
<td>1.89 (.757)</td>
<td>22.55 (.312)</td>
<td>17.93 (.592)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
<td></td>
<td>52.51 (.089)</td>
<td>34.28 (.932)</td>
<td></td>
<td>36.56 (.626)</td>
<td>45.81 (.244)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td></td>
<td></td>
<td></td>
<td>23.75 (.783)</td>
<td></td>
<td>31.33 (.178)</td>
<td>20.67 (.711)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.37 (.968)</td>
<td>3.21 (.782)</td>
<td>25.12 (.719)</td>
<td>24.95 (.728)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22.17** (.000)</td>
<td>6.23 (.285)</td>
<td>7.49 (.187)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.34 (.800)</td>
<td>6.31 (.277)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28.43 (.289)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key: $\chi^2$ statistic ; (p-value)
* $\chi^2$ statistic is significant at 5% level of significance
** $\chi^2$ statistic is significant at 1% level of significance
Table E.54: Analysis of variance (ANOVA) results for Q9, Q10 and Q15 to Q23 by the single response categorical variables.

<table>
<thead>
<tr>
<th></th>
<th>Q9</th>
<th>Q10</th>
<th>Q15</th>
<th>Q16</th>
<th>Q17</th>
<th>Q18</th>
<th>Q19</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>0.01</td>
<td>3.33*</td>
<td>2.02</td>
<td>2.03</td>
<td>1.59</td>
<td>2.51</td>
<td>1.75</td>
<td>1.75</td>
<td>0.17</td>
<td>0.80</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>(.990)</td>
<td>(.045)</td>
<td>(.974)</td>
<td>(.814)</td>
<td>(.217)</td>
<td>(.094)</td>
<td>(.842)</td>
<td>(.458)</td>
<td>(.989)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>0.14</td>
<td>0.74</td>
<td>0.13</td>
<td>0.36</td>
<td>3.76*</td>
<td>0.84</td>
<td>0.44</td>
<td>0.64</td>
<td>1.10</td>
<td>0.44</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>(.967)</td>
<td>(.573)</td>
<td>(.969)</td>
<td>(.835)</td>
<td>(.011)</td>
<td>(.510)</td>
<td>(.781)</td>
<td>(.636)</td>
<td>(.371)</td>
<td>(.777)</td>
<td>(.660)</td>
</tr>
<tr>
<td>Q5</td>
<td>1.04</td>
<td>0.95</td>
<td>1.87</td>
<td>0.84</td>
<td>2.93*</td>
<td>1.01</td>
<td>0.28</td>
<td>0.55</td>
<td>2.99*</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.400)</td>
<td>(.448)</td>
<td>(.135)</td>
<td>(.506)</td>
<td>(.033)</td>
<td>(.412)</td>
<td>(.887)</td>
<td>(.699)</td>
<td>(.030)</td>
<td>(.335)</td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>1.09</td>
<td>0.60</td>
<td>0.84</td>
<td>0.50</td>
<td>1.03</td>
<td>1.09</td>
<td>0.52</td>
<td>1.19</td>
<td>0.51</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.389)</td>
<td>(.731)</td>
<td>(.549)</td>
<td>(.801)</td>
<td>(.423)</td>
<td>(.386)</td>
<td>(.787)</td>
<td>(.335)</td>
<td>(.612)</td>
<td>(.709)</td>
<td></td>
</tr>
</tbody>
</table>

Key: F-statistic (p-value)

Table E.55: Summary table of Chi-Square test results for testing pairwise independence between two Likert scale variables, where (the row and column variables are Likert scaled).

<table>
<thead>
<tr>
<th></th>
<th>Q10</th>
<th>Q15</th>
<th>Q16</th>
<th>Q17</th>
<th>Q18</th>
<th>Q19</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q9</td>
<td>20.01</td>
<td>9.24</td>
<td>6.19</td>
<td>3.64</td>
<td>4.84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.067)</td>
<td>(.682)</td>
<td>(.402)</td>
<td>(.726)</td>
<td>(.848)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q10</td>
<td>12.71</td>
<td>30.52*</td>
<td>23.05*</td>
<td>11.26</td>
<td>31.95**</td>
<td>28.40**</td>
<td>16.57</td>
<td>60.10**</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.694)</td>
<td>(.015)</td>
<td>(.027)</td>
<td>(.187)</td>
<td>(.001)</td>
<td>(.005)</td>
<td>(.166)</td>
<td>(.000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.021)</td>
<td>(.256)</td>
<td>(.271)</td>
<td>(.294)</td>
<td>(.271)</td>
<td>(.765)</td>
<td>(.345)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q17</td>
<td>15.26</td>
<td>8.76</td>
<td>6.31</td>
<td>42.25**</td>
<td>38.94**</td>
<td>23.54*</td>
<td>24.62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.054)</td>
<td>(.723)</td>
<td>(.613)</td>
<td>(.000)</td>
<td>(.000)</td>
<td>(.023)</td>
<td>(.077)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q18</td>
<td>8.80</td>
<td>15.48</td>
<td>6.89</td>
<td>19.85*</td>
<td>18.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.185)</td>
<td>(.079)</td>
<td>(.648)</td>
<td>(.019)</td>
<td>(.109)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q19</td>
<td>7.53</td>
<td>8.10</td>
<td>3.29</td>
<td>8.51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.274)</td>
<td>(.231)</td>
<td>(.772)</td>
<td>(.385)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35.77**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(.000)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>28.57**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(.005)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(.397)</td>
<td></td>
</tr>
</tbody>
</table>

Key: $\chi^2$ statistic ; (p-value)
* $\chi^2$ statistic is significant at 5% level of significance
** $\chi^2$ statistic is significant at 1% level of significance
Table E.56: Summary table of Chi-Square test results for testing pairwise independence between a categorical variable and a Likert scale variable (row variables are categorical and column variables are Likert scaled).

<table>
<thead>
<tr>
<th></th>
<th>Q9</th>
<th>Q10</th>
<th>Q15</th>
<th>Q16</th>
<th>Q17</th>
<th>Q18</th>
<th>Q19</th>
<th>Q20</th>
<th>Q21</th>
<th>Q22</th>
<th>Q23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>30.99</td>
<td>38.03</td>
<td>19.42</td>
<td><strong>52.96</strong>*</td>
<td>20.36</td>
<td>12.99</td>
<td><strong>37.68</strong>*</td>
<td>51.69</td>
<td>30.84</td>
<td>39.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.154)</td>
<td>(.214)</td>
<td>(.248)</td>
<td>(.111)</td>
<td>(.676)</td>
<td>(.674)</td>
<td>(.037)</td>
<td>(.001)</td>
<td>(.159)</td>
<td>(.160)</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>1.31</td>
<td>7.47</td>
<td>4.13</td>
<td>6.93</td>
<td>8.26</td>
<td>7.91</td>
<td>5.52</td>
<td>5.80</td>
<td>9.32</td>
<td>4.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.971)</td>
<td>(.487)</td>
<td>(.389)</td>
<td>(.544)</td>
<td>(.220)</td>
<td>(.095)</td>
<td>(.479)</td>
<td>(.446)</td>
<td>(.157)</td>
<td>(.805)</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>7.65</td>
<td>19.23</td>
<td>4.16</td>
<td><strong>33.12</strong>*</td>
<td>9.14</td>
<td>5.22</td>
<td>16.54</td>
<td>13.82</td>
<td>7.80</td>
<td>24.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.812)</td>
<td>(.257)</td>
<td>(.842)</td>
<td>(.007)</td>
<td>(.691)</td>
<td>(.734)</td>
<td>(.168)</td>
<td>(.312)</td>
<td>(.800)</td>
<td>(.088)</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>2.90</td>
<td>1.70</td>
<td>3.03</td>
<td>1.58</td>
<td>1.95</td>
<td>0.81</td>
<td>1.44</td>
<td>2.73</td>
<td>1.45</td>
<td>5.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.407)</td>
<td>(.792)</td>
<td>(.220)</td>
<td>(.813)</td>
<td>(.584)</td>
<td>(.667)</td>
<td>(.697)</td>
<td>(.435)</td>
<td>(.694)</td>
<td>(.201)</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td><strong>24.20</strong>*</td>
<td>20.35</td>
<td>14.31</td>
<td><strong>26.30</strong>*</td>
<td><strong>28.59</strong>*</td>
<td>10.01</td>
<td>16.10</td>
<td>8.78</td>
<td>16.41</td>
<td><strong>39.00</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.019)</td>
<td>(.205)</td>
<td>(.074)</td>
<td>(.049)</td>
<td>(.005)</td>
<td>(.264)</td>
<td>(.187)</td>
<td>(.722)</td>
<td>(.173)</td>
<td>(.001)</td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>28.72</td>
<td>22.34</td>
<td>19.38</td>
<td>24.62</td>
<td>23.04</td>
<td>15.38</td>
<td>26.92</td>
<td>14.03</td>
<td><strong>38.56</strong>*</td>
<td>33.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.231)</td>
<td>(.898)</td>
<td>(.250)</td>
<td>(.821)</td>
<td>(.517)</td>
<td>(.497)</td>
<td>(.308)</td>
<td>(.946)</td>
<td>(.030)</td>
<td>(.383)</td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td>12.70</td>
<td>20.81</td>
<td>9.98</td>
<td><strong>36.32</strong>*</td>
<td>8.60</td>
<td>7.18</td>
<td>22.74</td>
<td>18.80</td>
<td>12.84</td>
<td>19.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.626)</td>
<td>(.408)</td>
<td>(.442)</td>
<td>(.014)</td>
<td>(.897)</td>
<td>(.708)</td>
<td>(.090)</td>
<td>(.223)</td>
<td>(.614)</td>
<td>(.514)</td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>10.48</td>
<td>21.30</td>
<td>7.87</td>
<td>10.74</td>
<td>16.28</td>
<td>9.00</td>
<td>8.93</td>
<td>18.97</td>
<td>11.87</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.915)</td>
<td>(.621)</td>
<td>(.795)</td>
<td>(.991)</td>
<td>(.573)</td>
<td>(.703)</td>
<td>(.961)</td>
<td>(.394)</td>
<td>(.85)</td>
<td>(.752)</td>
<td></td>
</tr>
<tr>
<td>Q11</td>
<td>5.86</td>
<td>7.50</td>
<td>2.75</td>
<td><strong>9.55</strong>*</td>
<td>4.37</td>
<td>1.84</td>
<td>7.22</td>
<td><strong>8.05</strong>*</td>
<td><strong>8.37</strong>*</td>
<td>6.65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.119)</td>
<td>(.112)</td>
<td>(.600)</td>
<td>(.049)</td>
<td>(.224)</td>
<td>(.398)</td>
<td>(.065)</td>
<td>(.045)</td>
<td>(.039)</td>
<td>(.155)</td>
<td></td>
</tr>
<tr>
<td>Q12</td>
<td>3.25</td>
<td>8.29</td>
<td>0.43</td>
<td>1.01</td>
<td>8.55</td>
<td>2.04</td>
<td>9.99</td>
<td>6.67</td>
<td>0.59</td>
<td><strong>9.48</strong>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.355)</td>
<td>(.082)</td>
<td>(.980)</td>
<td>(.605)</td>
<td>(.073)</td>
<td>(.563)</td>
<td>(.019)</td>
<td>(.083)</td>
<td>(.746)</td>
<td>(.050)</td>
<td></td>
</tr>
<tr>
<td>Q13</td>
<td><strong>24.35</strong></td>
<td><strong>34.36</strong></td>
<td>22.44</td>
<td>2.61</td>
<td>27.99</td>
<td>18.21</td>
<td>15.24</td>
<td>14.07</td>
<td>15.68</td>
<td>30.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.059)</td>
<td>(.024)</td>
<td>(.317)</td>
<td>(.989)</td>
<td>(.110)</td>
<td>(.252)</td>
<td>(.435)</td>
<td>(.520)</td>
<td>(.404)</td>
<td>(.059)</td>
<td></td>
</tr>
<tr>
<td>Q14</td>
<td>19.32</td>
<td>16.52</td>
<td>10.36</td>
<td>7.00</td>
<td>20.62</td>
<td>20.97</td>
<td>7.51</td>
<td>19.08</td>
<td>10.50</td>
<td>14.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(.200)</td>
<td>(.684)</td>
<td>(.961)</td>
<td>(.725)</td>
<td>(.420)</td>
<td>(.138)</td>
<td>(.677)</td>
<td>(.210)</td>
<td>(.787)</td>
<td>(.464)</td>
<td></td>
</tr>
</tbody>
</table>

Key: $\chi^2$ statistic ; (p-value)

* $\chi^2$ statistic is significant at 5% level of significance

** $\chi^2$ statistic is significant at 1% level of significance
E.6 Supportive information for the Field study

This section represents any complementary data that could support the field study report, with discussed in Section 5.4.

Table E.57: Checklist questions, Organisation answers and feedback for: Site1

<table>
<thead>
<tr>
<th>#</th>
<th>Checklist Questions</th>
<th>Answers given by the organisation</th>
<th>Organisation Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is the architect and evaluation team (testers) available on site?</td>
<td>No - There is no focus on the architecture within the organisation during the development of systems. The current direction of the organisation’s large systems is developing database systems using Oracle (ERP).</td>
<td>There is a peer review before production of systems artefacts.</td>
</tr>
<tr>
<td>2</td>
<td>If there is an evaluation team, what is their main field of experience? (Architecture, design, programming, systems evaluation, or others)</td>
<td>Most of the people involved in the system development are programmers and some are designers. However, there are no architects.</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>Is there any new system, which is under development on site?</td>
<td>Rank Competition System, Using ERP Oracle Application.</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>What is the expected due-date of the system/systems under development?</td>
<td>2 Months</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>“Was modelling language used in existing systems or for those systems that are under development?”</td>
<td>No</td>
<td>OK</td>
</tr>
<tr>
<td>6</td>
<td>“What was the current systems’ evaluation method against quality attributes?”</td>
<td>Normally, an iterative functionality acceptance test carried out after implementation finished. Also, regarding database systems, they are using Toad tools to check database relationships and some problem diagnosis such as SQL problematic statements and performance issues. There are no evaluation methods or tools that have been used on the architectural level.</td>
<td>OK</td>
</tr>
<tr>
<td>7</td>
<td>For the systems under development, did they use modelling techniques or languages? If yes, in which life cycle stage were they used? What are the accepted artefacts?</td>
<td>There is no use for any modelling language such as UML, SysML, or diagrams (components and connectors) such as DFD, State diagrams, etc. Their main method is prototyping using Visio and Form-Designer to create forms, grids, reports, interface screens, etc.</td>
<td>OK</td>
</tr>
<tr>
<td>No.</td>
<td>Question</td>
<td>Response</td>
<td>Notation</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Were modelling techniques used to architect the system?&quot;</td>
<td>No - There is no architecture focus within the organisation during the development of systems.</td>
<td>OK</td>
</tr>
<tr>
<td>9</td>
<td>Is there any architecture evaluation method that has been used? If yes, how was it used?</td>
<td>No - There is no architecture focus within the organisation during the development of systems.</td>
<td>OK</td>
</tr>
<tr>
<td>10</td>
<td>Are there any tools that support modelling?</td>
<td>No, except prototyping tools mentioned above.</td>
<td>OK</td>
</tr>
<tr>
<td>11</td>
<td>&quot;Are there any tools that support modelling evaluation? &quot;</td>
<td>No, except Toad tool, mentioned above, which not supporting evaluation on architectural level.</td>
<td>OK</td>
</tr>
<tr>
<td>12</td>
<td>Are there any tools that support reverse engineering? If yes, what is the produced artefacts format?</td>
<td>No</td>
<td>OK</td>
</tr>
<tr>
<td>13</td>
<td>Is there an interface mechanism within existing systems to an external analysis tool?</td>
<td>No</td>
<td>OK</td>
</tr>
<tr>
<td>14</td>
<td>&quot;Was the architecture of existing systems, or systems under development, developed according to some standards?&quot;</td>
<td>No - There is no architecture focus within the organisation during the development of systems. Mainly, they use what Oracle uses as a standard, such as Oracle ERP.</td>
<td>OK</td>
</tr>
<tr>
<td>15</td>
<td>&quot;Is the organisation familiar with DoDAF, Artisan, and Math-works products?&quot;</td>
<td>No</td>
<td>OK</td>
</tr>
<tr>
<td>16</td>
<td>&quot;Are there any systems simulation tools that have been used? If yes, for what purposes? What are the tools? What is the input and output of the simulation?&quot;</td>
<td>No</td>
<td>OK</td>
</tr>
<tr>
<td>17</td>
<td>&quot;Are there any standards, methods, techniques, tools, and procedures for documenting the software development processes and artefacts? If yes, what are they?&quot;</td>
<td>No, documenting is very primitive. Also, it varies from developer to another, and from one system to another. Types of documentation artefacts are also different with no standard.</td>
<td>OK</td>
</tr>
</tbody>
</table>
Table E.58: Checklist questions, Organisation answers and feedback for: Site2

<table>
<thead>
<tr>
<th>#</th>
<th>Checklist Questions</th>
<th>Answers given by the organisation</th>
<th>Organisation Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is the architect and evaluation team (testers) available on site?</td>
<td>&quot;No - There is no architecture focus within the organisation during the development of systems. The current direction of the organisation’s large systems is developing database systems and web applications using Oracle (ERP). There are some mid-size applications, that have developed on site. Normal waterfall life cycle has been used. Mid-size systems used Visual Studio as a tool based on .NET framework.”</td>
<td>Ok</td>
</tr>
<tr>
<td>2</td>
<td>If there is an evaluation team, what is their main field of experience? (Architecture, design, programming, systems evaluation, or others)</td>
<td>No, they have been grouped when needed, their main proficiencies are programming and design.</td>
<td>Ok</td>
</tr>
<tr>
<td>3</td>
<td>Is there any new system which is under development on site?</td>
<td>&quot;Financial and management system, and Aircrafts maintenance and Inventory system using Oracle (ERP), as a web based application.”</td>
<td>Ok</td>
</tr>
<tr>
<td>4</td>
<td>What is the expected due-date of the system/systems under development?</td>
<td>No due-date has been given.</td>
<td>Ok</td>
</tr>
<tr>
<td>5</td>
<td>Was modelling language used in existing systems or for those systems that are under development?</td>
<td>No, but they used some modelling diagrams such as use cases and entity relationship during the development of on site applications.</td>
<td>Ok</td>
</tr>
<tr>
<td>6</td>
<td>&quot;What was the current systems’ evaluation method against quality attributes??”</td>
<td>Normally, an iterative work-through test is carried out after implementation is finished. Regarding the database systems, they are using Toad tools to check database relationships and some problem diagnosis such as SQL problematic statements and performance issues. There are no evaluation methods or tools that have been used on the architectural level.</td>
<td>Ok</td>
</tr>
<tr>
<td>7</td>
<td>For the systems under development, did they use modelling techniques or languages? If yes, in which life cycle stage were they used? What are the accepted artefacts?</td>
<td>There is no use for any modelling language such as UML, SysML, or diagrams (components and connectors) such as DFD, State diagrams, etc. Their main method is prototyping using Visio and Oracle tools to create forms, grids, reports, interface screens, etc.</td>
<td>Ok</td>
</tr>
<tr>
<td>#</td>
<td>Checklist Questions</td>
<td>Answers given by the organisation</td>
<td>Organisation Feedback</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Were modelling techniques used to architect the system?&quot;</td>
<td>NO - There is no architecture focus within the organisation during the development of systems.</td>
<td>Ok</td>
</tr>
<tr>
<td>9</td>
<td>Is there any architecture evaluation method that has been used? If yes, how was it used?</td>
<td>No - There is no architecture focus within the organisation during the development of systems.</td>
<td>Ok</td>
</tr>
<tr>
<td>10</td>
<td>Are there any tools that support modelling?</td>
<td>NO, except prototyping tools mentioned above or Visio.</td>
<td>Ok</td>
</tr>
<tr>
<td>11</td>
<td>&quot;Are there any tools that support modelling evaluation?&quot;</td>
<td>None on architectural level. However, they are using some of the built-in oracle tools to evaluate web application components relations.</td>
<td>Ok</td>
</tr>
<tr>
<td>12</td>
<td>Are there any tools that support reverse engineering? If yes, what is the produced artefacts format?</td>
<td>No</td>
<td>Ok</td>
</tr>
<tr>
<td>13</td>
<td>Is there an interface mechanism within existing systems to an external analysis tool?</td>
<td>No</td>
<td>Ok</td>
</tr>
<tr>
<td>14</td>
<td>&quot;Was the architecture of existing systems, or systems under development, developed according to some standards?&quot;</td>
<td>No - There is no architecture focus within the organisation during the development of systems. Mainly, they use Oracle ERP standard</td>
<td>Ok</td>
</tr>
<tr>
<td>15</td>
<td>&quot;Is the organisation familiar with DoDAF, Artisan, and Math-works products?&quot;</td>
<td>No</td>
<td>Ok</td>
</tr>
<tr>
<td>16</td>
<td>&quot;Are there any systems simulation tools that have been used? If yes, for what purposes? What are the tools? What is the input and output of the simulation?&quot;</td>
<td>No</td>
<td>Ok</td>
</tr>
<tr>
<td>17</td>
<td>&quot;Are there any standards, methods, techniques, tools, and procedures for documenting the software development processes and artefacts? If yes, what are they?&quot;</td>
<td>&quot;No, the documenting that has been used is very primitive, Also, it varies from developer to another, also, from one system to another. Documentation documents follow no standardisation schema.&quot;</td>
<td>Ok</td>
</tr>
</tbody>
</table>

Table E.59: Checklist questions, Organisation answers and feedback for: Site3
<table>
<thead>
<tr>
<th></th>
<th>Supportive Information for the Field Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is the architect and evaluation team (testers) available on site?</td>
</tr>
<tr>
<td>2</td>
<td>If there is an evaluation team, what is their main field of experience? (Architecture, design, programming, systems evaluation, or others)</td>
</tr>
<tr>
<td>3</td>
<td>Is there any new system which is under development on site?</td>
</tr>
<tr>
<td>4</td>
<td>What is the expected due-date of the system/systems under development?</td>
</tr>
<tr>
<td>5</td>
<td>Was modelling language used in existing systems or for those systems that are under development?</td>
</tr>
<tr>
<td>6</td>
<td>&quot;What was the current systems’ evaluation method against quality attributes?&quot;</td>
</tr>
<tr>
<td>7</td>
<td>For the systems under development, did they use modelling techniques or languages? If yes, in which life cycle stage were they used? What are the accepted artefacts?</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Were modelling techniques used to architect the system?&quot;</td>
</tr>
<tr>
<td>No.</td>
<td>Question</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>9</td>
<td>Is there any architecture evaluation method that has been used? If yes, how was it used?</td>
</tr>
<tr>
<td>10</td>
<td>Are there any tools that support modelling?</td>
</tr>
<tr>
<td>11</td>
<td>&quot;Are there any tools that support modelling evaluation?&quot;</td>
</tr>
<tr>
<td>12</td>
<td>Are there any tools that support reverse engineering? If yes, what is the produced artefacts format?</td>
</tr>
<tr>
<td>13</td>
<td>Is there an interface mechanism within existing systems to an external analysis tool?</td>
</tr>
<tr>
<td>14</td>
<td>&quot;Was the architecture of existing systems, or systems under development, developed according to some standards?&quot;</td>
</tr>
<tr>
<td>15</td>
<td>&quot;Is the organisation familiar with DoDAF, Artisan, and Math-works products?&quot;</td>
</tr>
<tr>
<td>16</td>
<td>&quot;Are there any systems simulation tools that have been used? If yes, for what purposes? What are the tools? What is the input and output of the simulation?&quot;</td>
</tr>
<tr>
<td>17</td>
<td>&quot;Are there any standards, methods, techniques, tools, and procedures for documenting the software development processes and artefacts? If yes, what are they?&quot;</td>
</tr>
</tbody>
</table>
Appendix

F

Complementary information for The RCS – Chapter 6

This appendix contains information that support Chapter 6.
F.1 RCS analysis - Supporting Figures

In this section I include few figures for Real-time Control System (RCS) architecture to support the analyses presented in Chapter 6. Also, the intention is to help in visualising the analysis and to facilitate the discussion for readers.

Figure F.1: Each layer of the system contains a number of RCS-nodes, each containing – Behaviour Generation (BG), World Model (WM), Sensory Processing (SP), and Value Judgement (VJ)-modules. The nodes are interconnected as a layer style, or lattice, through the communication system, *Meystel et al.* [2002].
Figure F.2: Hierarchical image and entity frames within Sensory Processing (SP) component, After Meystel et al. [2002].
Figure F.3: Snapshot for the RCS Node structure with SysML, using Artisan tool.
Figure F.4: Extracted RCS architecture for coupling evaluation

Figure F.5: Modified model from the extracted RCS architecture, for comparison purpose.
Appendix

G

Opportunity to improve SA through future-work thoughts

Maintaining architectural integrity is not easy. Architectural quality requires investment and discipline.
— Dr. Bill Curtis, Senior vice president and chief scientist at CAST

Mistrík et al. [2014, pp xxvi]

This appendix briefly introduces incomplete and possible concepts for (future work) that could improve SA and SAE. However, full description and development of both concepts is outside the scope of this thesis.

The two concepts were brighten-up during the improvement of Moreno et al. [2008] performance model that discussed in Chapter 6, which are:

1. The first pillar is to describe (in brief) the System Architecture Evaluation (SysAE) approach, which is a vision of developing a full product to evaluate system (hardware and software) architecture in the future. SysAE is an approach that comprised of four main components as explained in Section G.2. Whereas, the Software Architecture Quantitative Evaluation Framework (SAQEF)—(second point), is one main component of SysAE product. SysAE concept, is organised in a package diagram, as illustrated in Figure G.1.

2. In the second pillar, I describe the “SAQEF” concept, in which I included some models to demonstrate the applicability of the framework. This concept is briefly discussed in Section G.2

The explanation of any standards, tools, languages, frameworks, and profiles that are included within the description of both concepts (SysAE and SAQEF) is out of this thesis scope.

G.1 SysAE approach - Big picture - Long-term vision

The idea behind this section is to explain my viewpoint regarding system evaluation. Also, to encourage engineers/developers to be involved in the SysAE approach by developing their own components incrementally, which can be integrated into SysAE product in the future. The drivers behind this package structure are both separation of concerns, and future development.
This structure, should allows developers to create their own evaluation models, then plug them into the SysAE-Product; even if they were geographically distributed. This is a privilege to maximise reuse, interoperability, minimise coupling, and maximise the independence given to the development teams through a component-based approach with a common repository.

However, it is important for individual developers to specify the interfaces early on, in order for their models to be linked and work with the main evaluation model SysAE-Product. This should be done with flexibility in mind.

As illustrated in Figure G.1, at the highest level of the package structure, is the SysAE–Product component which includes common libraries, such as the measurement units for evaluation metrics (e.g. Time in Second, Number of Components (NOC), Number of Components (NOC), etc.)

Figure G.1: System Architecture Evaluation (SysAE-Product) structure, Using component base products approach.

The second level of the structure contains two product packages HardwareAE, and SoftwareAE, which contain architectural evaluation general requirements. This was necessary due to the differences between hardware and software measuring techniques.

The H-Family-Req and S-Family-Req packages, contain the high-level system (hardware/ software) general use cases, system context diagram, and system constraints.
G.2. SAQEF EVALUATION FRAMEWORK AND MODELS

Whereas, the third level is the Component package of four different packages that would be available in each product. Differentiation between quantitative and qualitative assessments for both hardware and software is essential, due to their evaluation methods distinguish. Each package feature is a self-contained unit, which would contain the specific requirements (Specific-Req), Architecture, Design, Implementation, and Testing packages. The specific requirements for each component are derived from, and extended from, common general (Family-Requirements) as specified in the (HardwareAE, and SoftwareAE) packages.

Software engineers would then develop subcomponents that related to the software component, and the hardware engineers develop and elaborate the hardware specific details. The developers/engineers should insure the consistency of their models, interdependencies between models, and their evaluation component should be able to integrate and run smoothly with SysAE–Product as a whole.

Each feature/evaluation model may be developed incrementally, with additional capabilities, which could be added on a regular basis. Also, each new evaluation model, should go through a component test procedure, before plugging it into the SysAE–Product, to ensure that everything works correctly, then an integration test after linking the model. Information will be shared between the developers by means of configuration Management process.

The main requirements for developing SysAE components are:

- All components in SysAE shall be described and developed with a standardised known description language.
- All the four main packages included in the Component package shall satisfy their own family requirements.
- All the four main packages included in the Component package shall have an integration mechanism to SysAE–Product as a whole.

At the end of the project, SysAE-Product will have a library of well-documented evaluation features, which could be used to evaluate several system architectures.

G.2 SAQEF evaluation framework and models

According to the brief description of the SysAE approach in the previous section G.1, SAQEF component should satisfy the general requirements of the SysAE product, and the software family requirements (S-Family-Req) package.

Thus, SAQEF main goal is to evaluate software systems on an architectural level, quantitatively. With this, the approach should be developed with a comprehensive and understandable notation, which permit Software Architecture Evaluation (SAE) by means of using tactics and metrics.

The key points of SAQEF description and development are:

- Its top-level specifications, which explained the standards, framework, its architecture, development guidelines that SAQEF should follow during its development and the main functions that SAQEF should perform.
- Its concept and methodology, including a brief description of the utilised languages, framework, tools, and the reasons behind their selection.
• Its structure, which explained the top-level structure of SAQEF concept, views, packages and their relationship with each other. This point also described SAQEF views and how they should be linked with their requirements to ensure that they are satisfied, traceable, and refined through the efficiency requirement model.
• Modelling metrics, whose concept was to utilise the parametric diagram (SysML profile) in modelling quantitative measurements through the system coupling assessment.
• Full implantation of the concept artefacts, and validation of them through a test producers.

G.2.1 SAQEF specifications

As an example, the specifications of the SAQEF framework are highlighted in a sample top level requirements diagram in Figure G.2. Overall SAQEF specification contains three requirements as follow:

1. Standard
2. Quality assessment
3. Interface

Whereas, the second level of requirements includes:

1. ISO-standard quality tree (Evaluating SA based on ISO quality classification);
2. Individual quality tree (Evaluating SA based on particular quality classification, which will be created by developers for a specific assessment);
3. Quantitative measures (Evaluating SA based software metrics, which can be used by both utility trees mentioned above), where each one has a textual description included in its propriety.

For example:

• **ISO-standard quality tree requirement include the statement**: SAQEF shall accommodate the international quality model standard ISO/IEC 25010, to evaluate software architecture.

• **Individual quality tree requirement include the statement**: SAQEF shall be able to accommodate any individually quality tree created by developers, to evaluate software architecture.

SAQEF requirements diagram will be expand, based upon the further and future research.

G.2.2 SAQEF Concept and Methodology

G.2.2.1 Brief description of the Solution

The SAQEF concept is driven by the knowledge and analysis for the SAE state-of-the-art. As a result, the approach focuses on standardisation and incremental development procedures. This can be achieved by utilising some of the existing and reliable standards, languages, and frameworks, ...etc; for example, (the The Unified Profile for DoDAF/MoDAF (UPDM) profile, SysML language, DoDAF framework, and the utilisation of sophisticated modelling tool called Artisan). Also, SAQEF concept should permits valuable ideas to be employed such as, tactics, metrics, and analytical theories. Therefore, formalisation of the approach will be high, which increases the readiness for automation and simulations.
The use of a good tool (e.g. Artisan) is important, due to some advantages that can be realised such as, the flexibility of using several profiles, checking consistency, integration in/out the models.

The architectural data for candidate architectures will be input to SAQEF for analysis. This should be done bearing in mind the analytical constraints and environmental challenges. Then SAQEF should analyse the input data based on its quantitative measures. The assessment result will appear as benchmarking characteristics that can have further analysis and visualisation, using integrated tools (e.g. MatLab, Sysim, and Simulink), which help in obtaining the analysis results in an understandable format that will help architects in making any necessary refinements. High operational concept of the evaluation approach is illustrated in Figures G.3 and G.4, using DoDAF operational view (OV-1a), which shows SAQEF as a component of SAE concept.

G.2.2.2 Brief description of utilised standards, profiles, language, and framework including some of SAQEF examples

This section explains the utilisation of several profiles by SAQEF. SAQEF model as shown in Figure G.5, which is a package diagram that represents the domain model. SysML profile must be employed by SAQEF package, in order to include SysML stereotypes. The model also, could employ some of the libraries information included within the SysML profile for measurement use. That is why it’s important to import any profile as indicated. The use of access relationship to DoDAF profile, is because it has been considered as a (private import) for some of its elements that are required to be used by SAQEF model, such as, the DoDAF views.

UPDM, is a domain meta model profile. It is an Object Management Group (OMG) standard, published in September 2009. It is a standardised approach of expressing (the Department of Defence Architecture Framework (DoDAF), and the Ministry of Defence Architecture Framework (MoDAF) artefacts),
using UML, and SysML modelling languages. The main objective of the UPDM approach is to leverage commercial standards to (DoDAF and MoDAF) frameworks and to offer support tool interoperability. Languages and frameworks included within the UPDM profile, are illustrated by the UPDM structure in Figure G.6.

**G.2.2.0.1 Reasons as to why I have chosen UPDM:** It supports:

- Extensibility to other Architecture Frameworks
- Representation of Architectural Patterns
- Security views
- Standardisation
- Quality assessment through domain specific modelling
- Custom views and viewpoints
- Enterprise and System Architecture
**SAQEF EVALUATION FRAMEWORK AND MODELS**

*Figure G.5: UPDM structure.*

**UPDM imports all SysML profile, which supports SAQEF evaluation approach, due to the following points:**

- **SysML parametric diagrams:** This facilitates the incorporation of engineering analysis with the models architecture. For instance, Security and performance parameters in an (system view) SV-7 can be captured in the equations’ parameters.
- **SysML blocks diagrams:** These represent the structural elements such as the system (artefacts) operational node configurations capability facilitating the use of item flows, flow ports, as well as, properties value with distributions and units.
- **SysML activity diagrams:** These are used for the purpose of supporting the uninterrupted flow of hierarchies’ activity, modelling, and offering support for superior flow block diagrams functionality.
- **SysML requirement diagrams:** This facilitates the text based requirements that ought to be traced and captured in other elements of the model through the use of relationships, such as satisfy, derive, verify and refine.
- **SysML view and viewpoint:** This facilitates the providence of the model multiple perspective, as well as, the management, control and information organising.

SAQEF approach will follow the guidelines of the *Architecture Description Standard* (IEEE–42010), during its architecture development processes, as described in SAQEF requirements in Figure G.2, unless a new standard is published.

*The use of IEEE – 42010 standard, during the development of SAQEF, could elevate its concept for the following reasons:*

- Helps to standardised the approach.
The points noted above limit some of the challenges and limitations that are existed within current SAE approaches, which are discussed earlier in Chapter 2, such as standardisations, tool support, and extensibility.

G.3 SAQEF model organisation and views

The overall structure of SAQEF components is illustrated through (packaging diagram) in Figure G.7. The Figure shows the structure of the model, the model elements are represented in packages format, and the relationships between them are identified.

As an example, Figure G.7, demonstrate the relationship between SAQEF views (Other-Views and...
Quality-Views) with the rest of the user model are explicitly expressed using the «import» relationship. Noting that SAQEF other views means capability, operational, system, services, and other needed views from DoDAF profile. The comments (green box), notes (yellow box) and constraints (blue box) have been utilised to make the model easy to understand. Also, its worth to note that with this configuration, the «view» packages contain no model elements of their own, and that changes to the model in other packages are automatically updated in any of SAQEF views. In addition, the structure shows dependency between SAQEF structural and behavioural packages. Also, presents the satisfaction relationship between SAQEF model and some of SAQEF requirements as example.

Furthermore, the library package SAE, contains two packages that have the value types needed for SAQEF evaluation methods. The ’SI Definitions’ package contains common value types which already exists in the SysML profile. The other package ‘SAQEF Specific Library’ is add in value types developed during SAQEF development due to the needs of specific measurements for the SA.

More elaborating example of SAQEF’ components relationships is illustrated in , through an efficiency assessment example, in order to prove the concept applicability.

Also, the SAQEF scope, purpose, vision, project information, context, tools, views, environment in all view one (AV-1) model, is illustrated (in brief) in Figure G.9.
Figure G.7: SAQEF Packaging structure—(using packages and views) - (Package diagram) – Artisan tool – The package 'SAQEF Model' illustrate how the model organised into packages that contain model elements
Figure G.8: SAQEF (Efficiency assessment) – More details with employment of DoDAF views - (Requirement diagram) – Artisan tool
Figure G.9: SAQEF-AV-1, used of DoDAF views - All view (AV-1) - Artisan tool
G.4 Architectural tactics and metrics

SAQEF approach is not about creating new tactics and metrics for evaluating software architecture; rather than to help any developers to define and describe their own architectural (tactics and metrics) and integrate them into SAQEF, to be tested and used. Furthermore, this thesis has discussed some proposed tactics and metrics in Chapter 2. The rest of this section will explain in brief what might make good metrics to evaluate software architecture quantitatively.

The five main criteria that help for developing a good metrics as reported by Jaquith [2007] are:

1. Contextually precise.
2. Always measured.
3. Inexpensive to gather.
4. Expressed by cardinal (no ordinal) numbers or percentage.
5. Expressed by at least one unit of measure.

From the knowledge gained by this research, I think that, the above five points are generally suitable due to the following explanation:

• Metrics should be relevant and specific, to help decision makers to reflect upon their results.
• Also, the metric should not allow subjective criteria and human judgement, which constraint and help at the same time different people to come up with the same occlusion for the metric result.
• Furthermore, most of the metrics takes time to calculate, starting from gathering needed data, transforming and organising it as needed, computing, then translate the result into understandable format. Thus, a final version of SAQEF implementation should help to make the above process much cheaper and faster, through automation.
• Transferring rating measure to numbers (e.g. high-low to 1-2) might be an inappropriate approach for a metric or quantitative measure. As example of this, Zayaraz [2010] quantitative method for evaluating SA, when he used the five point scale for quality attributes and the pair wise comparisons when he transferred all verbal judgement into numerical judgement (Equally preferred = 1 … Extremely preferred = 9).
• Good metric should be measured by one unit at least, two even better. A simple example of this is, a number of security defects in an application. Better metric is a number of security defect per 100 lines of code in the application, or by per layer in architectural level.

During SAQEF conceptual development all measurement units for hardware and software should be placed in the ‘model library’ and all information will be shared by SysAE components. Any new measurement units could be added to the library with no difficulties. An example of the measurement units and value types are illustrated in Figure G.10, which shows ‘package diagram’ that demonstrate some SAQEF specific value types and units, which could be used in SA evaluation process.

G.4.1 Applying metrics with the parametric diagram

One of the main important components in SAQEF framework is the utilisation of ‘parametric diagram’ that could transfer theoretical assessment method, into a working evaluation mechanism to assess SA. There are many architectural evaluation methods, equations that could be applied into architectural
Further work need to be performed, in order to prove the concept of SAQEF approach in general and its ability to provide developers with a vehicle that could help them to develop their measurement matrices and equations, integrate them easily to SAQEF models, test them, then use them over again then again.

An example of ‘parametric diagram’ utilisation, is represented, in order to illustrate the idea of transferring equation (System Coupling Equation) developed by Zayaraz [2010], into a parametric model within SAQEF framework. Its important before modelling equations by parametric diagram to identify all the variables within the equation, and to identify all its input and output. I used block definition diagram package to define the equation for System coupling measurement’ as illustrated in Figure G.11. All parameters, properties, constraints, value types and units are defined within the Figure. The ‘System Coupling Equation’ constraints block does have ‘SCp’ (coupling equation) as one of its properties to facilitate the parametric modelling.

By clicking on the ‘System Coupling Equation’ constraints block in Figure G.11, the Parametric Diagram in Figure G.12 will appear, because both Figures are connected. This parametric diagram expanded ‘System Coupling Equation - constraints block’, were all input, output, rational (green box), and constraint notes (blue box) are identified and linked together. Extracting architectural data, and visualising equations result are are recommended for further research.

G.5 Conclusion

In conclusion, this appendix has been reported to support Chapter 7 future work discussion. Two important pillars have been briefly discussed with some examples. Both concepts SysAE and SAQEF are parts of the future work plan.
Figure G.11: Constraints and parameters details for (System coupling measurement) in Block Definition Diagram format (bdd) - SAQEF profile - Artisan tool.
Figure G.12: Parametric Diagram – System coupling. The constraint block from Figure G.11 has been expanded in this Figure where its equation is the constraints property (yellow box), which is belong to its block. Constraint notes are used to explain the system coupling equation and expected output limits - SAQEF profile - Artisan tool.
Author Index

Ölveczky, P. C., 33
Abou-El-Fittouh, H., 158
Abowd, G., 64
Abrial, J.-R., 37
Adrion, W. R., 6
Albus, J., 18, 216, 219, 220, 222, 225, 229, 245, 248
Albus, J. S., 216, 218, 228
Alebrahim, A., 73
Alexander, C., 3, 15, 17, 61, 62, 97, 112, 114
Allen, R., 304
Alpert, S. R., 114
Alshaikh, Z., 112
Armour, F., 81, 235
Athan, A., 64
AV Sriharsha, A. R., 164, 180
Babar, M., xii, xxi, 17, 64, 82–84, 86, 233
Babar, M. A., 100, 102
Bachmann, F., 4, 134
Badreddin, O., 32
Bardram, J. E., xii, 85
Bart, J. T., 114
Becker, S., xii, 16, 59, 89–91, 232
Bendik, J., 39
Bengtsson, P., 72, 73
Blanchard, B. S., 7
Booch, G., 34
Bosch, J., 304
Boucka, N., 37
Boyer, R. S., 38
Breivold, H. P., 18
Brookes, S. D., 90
Brouwers, N., 164
Buede, D. M., 20
Buschmann, F., 34, 98, 99, 110, 112, 114, 130, 228, 232
Cabot, J., xvii, 44
Capilla, R., 112
Carroll, S., 75
Cheng H.C., L., 18, 100
Chidamber, S. R., 239
Chrissis, M. B., 206
Clements, P., 5, 17, 19, 34, 39, 64, 66–76, 98, 134, 163, 235
Cochran, W. G., 141, 144
Company, H. A. C., 28
Comyn-Wattiau, I., 112
Coplien, J. O., 112, 114
Dashofy, E. M., 304
Denscombe, M., 135, 141
Desharnais, J., 116
Dey, A. K., 112
Dick, J., 29
Dissaux, P., 37, 304
Dobrica, L., 64, 66, 73
Dogru, A. H., 73
Donohoe, P., 229
Ejiogu, L. O., 75, 408
Everitt, B. S., 144
Fairbanks, G., 112
Feiler, P., xvii, 40, 41
Fielding, R. T., 16, 19, 112
Finkelstein, R., xviii, 218, 221
Fitzgerald, J., 37
Flint, S., xix, 7, 8, 55, 301, 302
Foster, H., 36
Fowler, M., 37, 45, 49, 60, 91, 107, 112, 130, 233
France, R., 103
Freitas, R., 19, 85, 130
Frølund, S., 238

411
Futrell, R. T., 95, 116
Gamma, E., 98, 99, 110, 114, 119, 130
Garlan, D., 40, 41, 78, 87, 98, 100–102, 112, 120, 121, 304
Gilmore, S., 91
Glass, R. L., 7
Gogolla, M., 38
Grady, R. B., 94–96, 180, 408
Grassi, V., 91
Graça, H., 112
Greenfield, J., 60–63
Griffiths, M., 249
Grone, B., 113
Gruniske, L., 85, 238
He, J., 304
Hewitt, C., 37
Hilken, C., 32
Hoare, C. A. R., 37
Hoinville, G., 141
Hummel, O., 130
Iqbal, Z., 231
Ivkovic, I., 58
Jaquith, A., 75, 180, 407, 408
Jones, C. B., 38
Jung, S. K., 134
Kamal, A. W., 241
Kan, S., 75, 97, 180, 408
Kassab M. and El-Boussaidi, G., 181
Kasunic, M., xviii, 135
Kaufmann, M., 38
Kazman, R., xviii, 64, 67, 70–72, 234, 235
Kelly, J., 35
Kerhervé, B., 58
Kerievsky, J., 232
Kerzner, H. R., 8
Khwaja, S., 130, 159, 164
Kim, H.-k., 73
Kim, J. S., 19, 53, 63, 88, 91, 130
Kircher, M., 110
Kitchenham, B., 135, 141
Klarlund, N., 36
Klassen, T. P., 20, 23
Klein, M., 79, 81
Kleppe, A., 50
Kleppe, A. G., 48, 54
Koziolek, H., 91
Krasner, H., 94–96
Kruegel, C., 84
Kurtev, I., 57, 58
Lamanca, B. P., 32
Lange, C. F. J., 62
Larman, C., 35
Lassing, N., 73
Leavens, G., 87
Leavens, G. T., 37
Lewis, W., 235
Linäker, J., 135
Lugou, F., 32, 164
Lung, C.-H., 73
Malavolta, I., 102, 158, 196
Manolescu, D., 63
Markovic, S., 58
Martin, J., 51
Mavridou, A., 41
Me, G., 146
Medvidovic, N., 32, 39, 102, 303, 304
Mello, R. M. de, 135
Mellor, S., 45, 47, 55
Mellor, S. J., xxii, 216, 223, 224
Merilinna, J., 56, 58, 76
Meystel, A. M., xviii, xix, 18, 216, 218–223, 225, 390, 391
Milicev, D., 76
Mistrík, I., 4, 53, 215, 395
Mitra, A., 116
Molter, G., 73
Mårtensson, F., 64
Noyer, A., 32
Obbink, H., 34, 64
Oquendo, F., 33, 40, 304, 306
Ortega, J. L., 220
Overhage, S., 18, 64
Ozkaya, M., 196
Pastor, O., 52–54
Pereira, V., 32
Petrasch, R., 116
Petri, C. A., 37
Petriu, D., 91
Pons, C., 44
Potts, C., 8
Pressman, R., 30, 31, 39
Punter, T., 56
Qin, Z., 4, 5, 17, 19, 30, 31, 33, 36, 38, 64, 66–68, 70, 73, 80, 302, 304, 306
Qumer, A., 51
Rech, J., xxi, 43–45, 56, 62, 91, 103, 306
Rensink, A., xxi, 46, 47
Reussner, R., 4, 81, 87, 125, 232
Riaz, M., 130
Ribeiro, F. G. C., 31, 32
Ribeiro, Q. A. D. S., 32, 164
Richards, M., 112
Riehle, D., 112, 113
Roscoe, A., 90
Rozier, K. Y., 31, 36
Runeson, P., 216
Rushby, J., xxi, 30, 38, 47, 191
Russell, D., 216
Röttger, S., 56, 58
Safwat, A., 29
Schlosser, H., 18
Schmidt, D., 61–63, 110
Schmidt, D. C., 63, 91, 120, 130
Schumacher, M., 59, 60, 62, 63, 110, 111, 115, 120, 239
Schumann, J. M., 35, 36, 39
SEI, 71, 235
Selman, B., 99
Shaw, M., 34, 41, 62, 303
Shreelekhya, G., 5
Silingas, D., 76
Silva Melo, M. da, 32
Silverston, L., 61
Simpson, J. A., 74
Sottet, J. S., 58
Stahl, T., 44, 45, 48, 49, 59, 60
Storey, N., 35
Suman, M., 116
Tang, A., 135, 141, 158, 196
Taylor, D. S., 38
Taylor, R. N., 304
Teebiga, R., 130
Tellis, W., 200
Tešanovic, A., xviii, 233
Tibermacine, C., 239
Völter, M., 62
Walker, P. M. B., 74
Wayne, W., 141
Weilkiens, T., 8
Wolfgang, P., 114
Wortmann, A., 32
Xia, S., 38
Yin, R., 200, 216
Zachman, J. A., 16
Zamansky, A., 38
Zayaraz, G., xii, 19, 67, 73, 74, 91, 130, 134, 164, 407, 408
Zhang, C., 158
Zhu, L., 134
Zimmermann, O., 63
Zou, Y., 58, 59