USE OF THESES

This copy is supplied for purposes of private study and research only. Passages from the thesis may not be copied or closely paraphrased without the written consent of the author.
Knowledge and Innovation

in Intellectual Property:
The Case of Computer Program Copyright

Gillian Catherine Dempsey  LLB (Hons)

A thesis submitted for the degree of Doctor of Philosophy
of The Australian National University.

March 1998
Sub-rule 15(2) Statement

I certify that this thesis is my own work and all sources used have been acknowledged.

Gillian M. Dempsey
Acknowledgements

This thesis is dedicated to Andrew Greinke, my husband, for the love and assistance which has seen the work through to completion.

Thanks to my supervisory panel: Dr Roger Clarke, Dr Peter Drahos, Prof. Don Lamberton and Dr Gregory E.P. Shailer for their organisation, cooperation and above all, for their patience in reading and commenting on this work through its evolutionary stages.

John Braithwaite and Anthony Wensley have my gratitude for delivering timely advice and moral support with their usual good humour.
Abstract

Information economics is used to develop a model of technological innovation which is applied to the case of computer program copyright. A critical outline of the neo-classical economic perspective of innovation and Arrow’s concerns regarding appropriability of information is provided. This perspective justifies intellectual property institutions as a correction of market failure and as a “reward for invention”. The same literature marginalises countervailing arguments including monopoly distortions, alternative sources of innovator reward and the potential for anti-competitive strategies.

Information economics provides a distinct and preferred perspective in the analysis of technological development and in the role of intellectual property in the promotion of innovation. The conception of information as a resource, rather than as a commodity, implies that information is part of a shared technological capital, whose indivisibilities should be exploited for social benefit. The information perspective conceives innovation as a messy, evolutionary and interactive process involving many participants, and a cycle of innovation characterised by incremental improvements, imitation and learning strategies, and technological trajectories influenced by bounded rationality. These environments will also generate powerful network externalities.

A model of innovation based on these assumptions is developed which incorporates two major distinctions. One is between tacit and codified knowledge; the other is between technology and technological artefacts. This knowledge-artefact distinction is defined in the innovation model by the concept of an information technology artefact, characterised as a physical product whose underlying means of creation is not communicated by mere possession of that product. This innovation model is reconciled to the intellectual property regimes of confidential
information, patent and copyright, demonstrating the use of legal
doctrines to encourage the diffusion of tacit knowledge through society.

Applying the innovation model to the question of computer programs, it
is argued that programs in their executable of machine code forms
 correspond to the concept of an IT artefact, in that possession of machine
code does not imply access to the underlying source code. The process of
software development and the utility of decompilation are discussed in
this context, particularly the lack of isomorphic correspondence between
machine code and third or higher generation source code languages. The
close analogy between the software development model and the scenario
of confidential information suggests a limited role for copyright of
computer programs beyond a prohibition of literal copying or piracy.

Arguments favouring broader protection of non-literal elements of
computer programs are critically reviewed and prescriptions for
proprietary protocols, user interfaces and standards in the literature are
rejected as inconsistent with the realisation of network externalities by the
software industry. An information economics perspective instead
recommends the encouragement of reverse engineering and imitative
competition provided that developers implement their own source code
solutions to invest in the diffusion of tacit programming knowledge.
Decompilation should be permitted to provide a limited degree of access
to internal interfaces and communications protocols. Elements of a user
interface should not be protected. Copyright regimes in the United States,
Europe and Australia are assessed against the policy prescriptions
generated by the application of the innovation model to computer
programs. The influence of political actors and international pressures
such as TRIPS are noted. It is hoped that the infusion of an information
economics approach might trigger the switch in perspective needed in
policy debates to preserve the integrity of the intellectual commons.
# Contents

CHAPTER ONE
INTRODUCTION ........................................................................... 1
1.1 OVERVIEW ........................................................................ 1
1.2 MOTIVATION AND SCOPE .............................................. 2
1.3 STRUCTURE ....................................................................... 5

CHAPTER TWO
INFORMATION AS A COMMODITY .............................................. 10
2.1 INTRODUCTION .................................................................. 10
2.2 ARROW’S MODEL OF INFORMATION .................................. 12
2.3 REWARD FOR INVENTION .................................................. 16
2.4 MONOPOLY EFFECTS .......................................................... 20
2.5 UNDER-REWARD AND OVER-REWARD ............................... 25
2.6 SUMMARY ........................................................................ 32

CHAPTER THREE
INFORMATION AS A RESOURCE ............................................... 34
3.1 INTRODUCTION .................................................................. 34
3.2 INFORMATION PROPERTIES ................................................. 36
3.2.1 Information is capital ....................................................... 37
3.2.2 Information is shared ....................................................... 38
3.2.3 Innovation is imitation ..................................................... 40
3.3 INFORMATION AND MARKETS ......................................... 43
3.3.1 Lead time ..................................................................... 44
3.3.2 Bounded rationality ........................................................ 46
3.3.3 Network externalities ....................................................... 48
7.6 CURRENT CASE LAW ................................................................. 188
7.6.1 United States ..................................................................... 189
7.6.2 Europe ............................................................................. 198
7.6.3 Australia .......................................................................... 200
7.7 POLITICAL CONSIDERATIONS ........................................... 207
7.8 SUMMARY ........................................................................... 215

CHAPTER EIGHT
CONCLUSIONS ........................................................................... 220
8.1 REVIEW ............................................................................... 220
8.2 CONTRIBUTIONS ................................................................... 225

APPENDIX ONE – THE COMPILATION PROCESS ...................... 229

APPENDIX TWO – COMMENTED SOURCE CODE ...................... 236

APPENDIX THREE – DECOMPILATION ..................................... 238

APPENDIX FOUR – APPLICATION FRAMEWORKS ................... 240

REFERENCES ............................................................................. 244
Chapter One
Introduction

1.1 Overview

Economic activity is increasingly dominated by the production, distribution and consumption of information. Developments in the literature herald a new age of rapidly expanding “knowledge industries” and “information economies”.¹ Computer programs are increasingly important facilitators of the use of technology, in applications as diverse as word processing, robot-based manufacturing and medical science.

A key issue in this information economy is the role of intellectual property in promoting technological innovation. A tension exists between offering a reward of a “limited monopoly” as an incentive to innovators, and preventing that limited monopoly from both exacting excessive profits and discouraging or impeding the production of information by limiting later author’s access to the public domain. The policy question is where to strike the proper balance between these objectives in the interests of innovation and technological advancement.

This thesis uses an information economics perspective to develop a

model of technological innovation which is the applied to the case of computer program copyright to suggest where this balance should be found.

1.2 Motivation and scope

Much of the legal literature in the area of copyright and computer programs is concerned with the extent to which the law does or does not apply to program code, to its "structure, sequence and organisation", and to other "non-literal" elements such as user interfaces. Few articles have engaged in a normative evaluation of desirable characteristics for an intellectual property regime for computer programs.² This thesis seeks to provide such a contribution through the development of a model of innovation grounded in information economics. Statements by legislators, the judiciary and commentators in respect of contentious issues of the practical interpretation of software copyright legislation demonstrate an attachment to neo-classical economic theory and faith in the application of legal semantics such as the idea-expression dichotomy. This thesis challenges both these approaches.

Legal systems derived from English jurisprudence have traditionally been seen as reluctant to extend property protection to the products of

"ingenuity, knowledge, skill or labour". The granting of intellectual property rights has rested largely on the economic argument that they provide stimulation to innovation, thereby promoting national economic advancement and consumer and social welfare. Economics is, therefore, used as a benchmark against which the policy recommendations developed in this thesis, and those submitted by others in the literature, can be assessed. Like Machlup, the thesis assumes that:

While economic analysis does not yet provide a basis for choosing between "all or nothing", it does provide a sufficiently firm basis for decisions about "a little more or a little less" of various ingredients [of intellectual property].

More particularly, however, an information economics approach is adopted. Information economics is the area of economics which analyses the "processes by which information and knowledge is produced, diffused, stored and used". The information perspective assumes that information capacities of economic actors are imperfect due to information asymmetries, arising from literacy, geographical location and technical proficiency, as well as degrees of cognition, based on language, innate intellectual ability and education. The analysis presented within

---

3 e.g. *Victoria Park Racing and Recreation Grounds Co. Ltd v Taylor* (1937) 58 CLR 479 at p.509 per Dixon J.


this thesis falls within the fourth strand of Machlup’s list of areas comprising information economics:

\[ \ldots \text{analysing the creation and utilisation of new technology, and the incentives for research, development, invention, innovation and diffusion of knowledge.} \]

Computer programs are themselves special forms of technology and it has been characteristics such as ease of replication which has driven legislative action to include computer programs as “literary works” within copyright laws internationally. Some assertions within the literature, however, of easy access to the knowledge underlying a computer program require reappraisal. Close attention to the nature of computer programs and the software development process is, therefore, essential to challenge these assumptions.

Likewise, it is empty to argue about the economic value of an intellectual property system without specifying and evaluating that system’s criteria. Legal analysis is, therefore, required of the copyright regimes in place internationally to assess whether the current state of the law is consistent with the policy recommendations developed from the innovation model.

The scope of the thesis includes three major foci for discussion: an economic analysis and development of the model of innovation; the

---


application of this model to software through a consideration of computer
program qualities; and finally an assessment of the state of copyright law
internationally.

1.3 Structure

Chapter Two notes that the foundation on which arguments for broader
protection of intellectual property and computer programs copyright in
particular, are based within a “reward for invention” argument. This
argument has its origins in work by Arrow and concerns about the
appropriability of information. Limiting access to information through
the imposition of intellectual property rights seeks to mimic the
excludable and tradeable qualities of physical goods. The chapter also
demonstrates that later literature has extrapolated this argument to the
point where the “balance” issue – public domain access to information –
is marginalised. Most particularly this is seen in downplaying the
potential distortions from monopoly rights, the avenues for rewarding
innovators other than through royalties, and the potential use of
intellectual property for anti-competitive strategies.

Chapter Three provides a counterpoint to the neo-classical “commodity”
view of information in outlining the preferable view that information is
instead a “resource”, or an element of a shared technological capital. The
characteristics of information and the innovation process are contrasted
between the neo-classical and information economic perspectives. The
neo-classical view is progress through “invention”, an isolated process
typically involving a single firm. The information economics view is that progress is an evolutionary and interactive learning process involving many participants. This process suggests a cumulative and incremental cycle of innovation driven by bounded rationality and displaying powerful network externalities. Such an environment implies significant lead times providing rewards to innovation.

Chapter Four uses an information economics perspective to develop a model of innovation. In this model, information does not possess strong "public good" characteristics, but its flows are limited by its embodiment in modes of human understanding and within its application to production. Two key concepts are drawn from existing literature and combined in this model. One is a distinction between "tacit" and "codified" knowledge, while the other is a distinction between knowledge in abstract and knowledge embodied in a physical product, defined as an "information technology artefact" or "IT artefact" for the purposes of this thesis. A significant assumption in this IT artefact model of innovation is that possession of the artefact (the physical product) does not communicate the underlying knowledge or means for its creation (the tacit and codified knowledge). This model is applied (and adapted where appropriate) to the intellectual property regimes of confidential information, patent and copyright. It is argued that the intellectual property laws promote innovation by the encouragement of flows of tacit knowledge through society.
Chapter Five applies the IT artefact model of innovation to computer programs. This involves a demonstration of the central quality of an IT artefact (program in executable form) – lack of access to underlying knowledge (source code and materials) – thereby, a rejection of a widely held assumption within the legal literature. This is based on an examination of the nature of computer programs and the development process, supplemented by the Appendices. In particular the chapter argues that third and higher generation languages bear no isomorphic identity with their compiled machine code and that decompilation (more accurately disassembly) is of limited use in understanding the source code of a program of any size or sophistication. It is argued that a closer analogy exists between the IT artefact model of computer program innovation and the regime of confidential information than to the regime of copyright, which assumes a high degree of access to codified knowledge from possession of the product. This implies that progress is by imitative competition and reverse engineering, with a limited role for copyright in prohibiting replication or “piracy” of machine code programs.

Chapter Six critically reviews the legal and economic literature favouring broader protection of computer programs than the limited role suggested by the innovation model. In particular the issue is whether copyright should extend to the more abstract, creative and stylistic elements of software, so called “non-literal” copying. Economic arguments are found to be based on a neo-classical perspective and assume maximisation to
returns to innovators and protection of investment as the overriding objective. Creativity arguments misconceive the nature of the computer programming as akin to a literary or artistic endeavour. The realisation of network externalities (and avoidance of their anti-competitive exploitation) dictates that interfaces, including user interfaces, and standard programming techniques should not be the subject of proprietary claims in copyright. Policy recommendations are summarised at the conclusion of this chapter.

Chapter Seven examines the United States, Europe and Australia to assess the state of the legislative and judicial interpretations of copyright over computer programs against the policy prescriptions suggested by the information perspective. The focus of discussion is whether the idea-expression distinction and the doctrine of fair use (or "fair dealing") are effective mechanisms in drawing the line between protected and unprotected elements of programs, and in providing a "bright-line" test for infringement. Extensions of the concept of "expression" to include "non-literal" aspects of programs – at odds with these prescriptions – can be reconciled as a confusion between evidentiary and doctrinal issues. In particular, the approach of the United States Second Circuit in *Altai*\(^9\), involving a complex factual determination, reflects a methodology based on circumstantial evidence of copying of source code. The preferred approach where there is no evidence of misappropriation in access to

source code or related materials is articulated in the appeal judgments in Borland\textsuperscript{10} (United States, First Circuit) and Powerflex\textsuperscript{11} (Australia, Full Federal Court). This approach is based on whether the non-literal element is itself protectable under copyright. The state of the law will, however, remain uncertain until the US Supreme Court is able to rule on a new appeal\textsuperscript{12} and the Australian High Court hears the pending appeal in Powerflex. The political considerations surrounding intellectual property policy are also discussed, with particular emphasis on coalitions of vested interests, international mechanisms such as the Berne Convention and Trade Related Aspects of Intellectual Property Rights (TRIPS), and unilateral trade actions by the United States Government in driving change in the direction of broader and longer protection.

Chapter Eight provides a overview of the discussion and its major findings, outlines the contributions made by this thesis, and concludes with a reflection related future issues.

\textsuperscript{10} Lotus Development Corporation \textit{v} Borland International Inc. 49 F. 3d 807 (1st Cir. 1995).

\textsuperscript{11} Powerflex Services Pty Ltd \textit{v} Data Access Corporation [1997] FCA 490.

\textsuperscript{12} The Borland appeal was tied 4-4 so the decision of the First Circuit stood.
Chapter Two

Information as a Commodity

2.1 Introduction

The major objective of this thesis is to examine innovation from an information economics perspective and to examine computer program copyright in that context. This chapter provides a review of the orthodox economics literature against which such an information economics perspective can be juxtaposed. Many policy arguments in respect of copyright, and protection of computer programs in particular, rely on the economic orthodoxy critically examined in this chapter. The review begins with the seminal work of Arrow and demonstrates its extension to more general propositions about intellectual property policy. The central feature of the orthodox economic argument is the maximisation of reward to the inventor. At the same time other opportunities for reward, the distorting effects of monopoly, and avenues for market power have been marginalised within the literature.

The policy objective of intellectual property is, in broad terms, the encouragement of technological progress.\(^\text{13}\) Conventional economic analyses suggest that imposition of “commodity” characteristics upon information will overcome a perceived lack of appropriability of the

---

information thereby securing to the creator the chance of a return which encourages investment. The traditionally accepted model of invention within the economics literature has been one where information is seen as the result of a process analogous to industrial output. Nonexcludable and nonrivalrous characteristics of this “invention-product” are said to justify enforcement of property rights on the basis of “market failure”. The key propositions, elaborated below, of the conventional model are as follows:

- without regulation, information has “public good” characteristics;
- lack of appropriability associated with public goods leads to market failure and underinvestment in inventive activity;
- through artificial scarcity imposed by intellectual property laws, the value of information in the hands of users becomes appropriable;
- investment in the creation of information is directly related to the degree to which this value can be appropriated by the investor/owner.

The above propositions form the basis for much of the economic literature concerned with intellectual property policy. As later discussion demonstrates\(^\text{14}\) such economic arguments are widespread within the computer program copyright debate. From an information perspective, however, such arguments may justify inappropriate public policy choices.

\(^{14}\) See particularly Chapter Six.
The role of this chapter is to provide a critical overview of the foundations of this literature against which the information perspective is a counterpoint.

2.2 Arrow’s model of information

The pedigree of most of the neo-classical economic literature on intellectual property can be traced from Arrow’s 1962 paper\(^\text{15}\) on the allocation of resources for invention. At this point it is appropriate to distinguish two terms used in this thesis. The term “invention” is used in this chapter since it is found in most the economic literature. The remainder of this thesis uses the term “innovation” to describe the process of technological progress in abstract.\(^\text{16}\) The concept of invention as a “product” and connotations of intra-firm technological development are consistent with the orthodox literature presented in this chapter. The term “innovation” is based on an information perspective which emphasises the incremental nature of the process and is chosen deliberately in contrast to the term “invention” which suggests an isolated event of inspiration. While the connotations differ, the denotation of both “invention” and “innovation” is technological progress in general. The terms appear to be used interchangeably within

---


Arrow discussed information problems in respect of agents within a market. The three key propositions that neo-classical economic theory has borrowed from Arrow are that:

- uncertainty leads to underinvestment in risky activities (such as invention);
- information is nonexcludable, which leads to market failure; and
- information is indivisible: the cost of producing information is unrelated to its value to users.

Arrow found that uncertainty creates risk and that devices for shifting risks are limited and imperfect (including, but not limited to, moral hazard effects).\(^{17}\) According to his analysis, there is an expectation of underinvestment in risky activities relative to a Pareto optimal allocation of resources to such projects. Arrow argued that, within a context of market uncertainty, information itself "becomes a commodity" since it is valuable to economic agents in overcoming uncertainty.\(^{18}\) Arrow characterised invention as the production of this information-commodity. Inherent uncertainties in production also suggest an underinvestment in inventive activities.

He identified a "fundamental paradox" of demand for information: that

---

17 *Ibidem* pp. 610-614.
18 *Id* p. 614.
Chapter Two - Information as a Commodity

the value to the purchaser is not known until the content is revealed, at which point the potential purchaser has acquired the information to some degree. Information is said to be "nonexcludable". Sale on an open market, therefore, may result in a purchaser reproducing the information at little or no cost. This is the origin of the frequently cited "appropriability" problem; the second important factor leading to an underinvestment in invention and research. This has also been described as the "public good" problem. A product is said to be a public good if those who consume the good might not pay for it (nonexcludability) and its benefit in consumption by one person does not unequivocally reduce its benefit in consumption by another (nonrivalrous competition).19

Arrow's third element was "indivisibility": that the cost of producing information is unrelated to the extent of its use and value in the hands of others. He noted that the cost involved in transmitting information is frequently very low. An efficient (Pareto optimal) market outcome would imply distribution of the information at marginal cost, which may be zero. Social welfare would, therefore, be maximised by diffusion of information throughout society at little cost. Pareto optimality is typically used as a benchmark for maximisation of economic welfare. It is the point at which no individual can be made better off without making another individual worse off. Arrow's analysis suggests that free

diffusion of information and knowledge throughout society promotes economic welfare.

A graphical representation of the Arrow model is provided in Figure 2-1. The purpose of the representation is to act as a prelude to the model implied by the conventional analysis (Figure 2-2) and the contrasting informational model of innovation (presented in Chapter Four). In the Arrow model the capital provider invests in a process of invention. Invention is a process which produces information (an intangible commodity). Information is valuable to users in managing risk, yet its full value is not appropriable to the capital provider since information may be (almost) costlessly transmitted to other users who do not provide a return ("free riders"). In evaluating the cost of investment to the benefits of returns, Arrow's model implies that there will be underinvestment relative to a measure of Pareto optimal efficiency. This proposition is the major focus of subsequent literature. The other proposition relating to indivisibilities, however, suggested that the value of the benefits of information production are unrelated to the value of investment. That is, some types of information may be produced at little cost that are of enormous benefit to society. The intellectual property literature has instead assumed a significant level of investment in research and development and limited avenues for extracting a return. This theme is developed in the following section.
Arrow's propositions:
1. Risk leads to underinvestment.
2. Nonexcludability creates market failure due to lack of appropriability.
3. Indivisibilities mean that value of benefits are unrelated to value of investment.
2.3 Reward for Invention

It is axiomatic within the orthodox literature that, once an information-good has passed beyond the inventor and any immediate contracting parties, it is difficult or impossible to recover from third parties a share of the benefits which flow to them from use of the information-good. The argument follows that, in the private market, there will be underproduction, relative to a "socially optimal" quantity and quality of information-goods, unless there is some government intervention to create an incentive to invest.

From this simple proposition comes the principal justification for intellectual property rights - the hypothesis that both risk and appropriability can be addressed (imperfectly according to Arrow) by creating "legally imposed property rights". Demsetz, in commenting directly on Arrow's paper, argued:

The degree to which knowledge is privately appropriable can be increased by raising the penalties for patent violations and by increasing resources for policing patent violations. . . . Given the appropriate legal apparatus and schedule of penalties it may be no more difficult to police property rights in many kinds of knowledge than it is to prevent the theft of automobiles and cash.

This argument by Demsetz was central to much of the policy literature and political arguments surrounding intellectual property. His assertion

---


was that maximisation of private appropriability through intervention of legal institutions (such as copyright) would facilitate market mechanisms and provide social benefit.

Much subsequent literature from this tradition - particularly that originating from the Chicago school\textsuperscript{22} - has accepted the "reward for invention" argument as flowing naturally from Arrow's paper. Private property rights are said to provide an \textit{incentive} for the production of intellectual property because, if successfully marketed, there is a chance of obtaining a return in the form of income which creators can obtain through royalties and licensing.\textsuperscript{23} The role of intellectual property laws, therefore, can be seen as to ensure the appropriability of the value of the works when in the hands of users. Failure to provide for appropriability in this fashion leads to a less than socially optimal amount of investment in invention.\textsuperscript{24}

The "reward for invention" thesis assumes that market failure is corrected only if intangibles are forced, through legal institutions, to exhibit the same economic characteristics as tangibles. The conception of information creation as a production chain is a necessary incident of the


conception of information as a commodity. The consequences of this commoditiy approach are portrayed in Figure 2-2. The purpose of this figure, prepared by the author, is to show the conceptual extension from Arrow’s model which the orthodox literature reflects. In particular a commodity model of information is analogous to the standard economic model of the market, with the process of “manufacture” replaced by “invention”. The product is not a physical good but an intangible, “information”, which is made artificially scarce due to the applications of intellectual property laws. Importantly, the perception is that invention is a single, isolated process which takes place within the “black box” of the firm. Due to legal sanctions and policing, the information can be marketed and its value in the hands of users appropriated through normal market mechanisms. The incentive to invent in such a model is directly related to the size of the potential return on investment.

Arguments which base their assumptions upon the concept of information as a commodity have typically made an implicit assumption of a direct relationship between the scope of the property right, the returns to inventors and the level of investment in research and development. This direct relationship drives the conclusion that the objectives of intellectual property can be promoted by maximisation of potential


Figure 2-2 - The Commodity Model

- Capital Provider
- Manufacture
- Good
- Consumers
- Process
- Product
- Market
- Invention
- Information
- Users

Incentive to Invent

- Natural scarcity
- Artificial scarcity through intellectual property laws
- Sale
- Licensing
- Investment
- Return
returns to inventors. It follows that activities such as price discrimination through licensing between individual and "site" software licences; hardback and paperback books; and individual and library subscription rates for journals, will serve to assure returns to investors by increasing the market power inherent in the property right.\(^{27}\)

This reasoning leads to the conclusion that broadening the scope of monopoly rights granted by intellectual property laws is always desirable, as is the limitation or abolition of measures such as compulsory licensing; reinforcing government policing and imposing higher penalties for infringement.

As an example, consider how the optimal life of an intellectual property right should be determined. The classic economic test in the area of determining a property right length was stated by Nordhaus:\(^{28}\)

As the life is increased, two opposite forces affect the level of economic welfare. First, a longer life increases invention and thus gives on balance a larger amount of output for a given level of inputs. This is a positive effect. Second, a longer life means that the monopoly on information lasts longer and thus there are more losses from inefficiencies associated with the monopoly. The optimal life of the patent is that point at which the two forces balance at the margin.

Many papers, however, appear to focus on Nordhaus's positive effect and


marginalise the negative effect. Kitch\textsuperscript{29} rhetorically noted that, viewed in isolation, the reward function of intellectual property suggested a perpetual term for patents:

If the purpose is to reward the inventor for his invention, then why shouldn’t he be awarded all of the present value of his invention?

Kitch also conceded, however, that the "simplicity" of the argument breaks down once the view is taken that the inventor’s contribution is not the invention itself, which eventually would have been made by someone else, but the time of the invention:\textsuperscript{30}

The patent should reward not for the whole value of the invention, but for the value of being first.

### 2.4 Monopoly effects

The focus on the “reward for invention” argument within the debate on duration is illustrative of the wider copyright literature which emphasises the incentive to invest yet downplays the potential market distortions associated with exclusive rights. This section examines the narrow approach taken to questions of monopoly within the intellectual property context.

There is a perception within the literature that there is small chance of intellectual property having undesirable economic effects. This relies to


\textsuperscript{30} \textit{Ibidem} p. 290.
some degree on the "paradox" of intellectual property protection: that the imposition of a monopoly privilege over the use or reproduction of an invention or work is alleged to correct market failure. Under the paradox, reconciling competition policy and protection of intellectual property appears to rest on a trade-off between short-run static inefficiency and long-term dynamic efficiency. Dynamic efficiency includes the invention and commercial introduction of new products and processes which enhance welfare both by increasing the utility of goods and by promoting growth through increased productive efficiency.

At one level, the negative impact of intellectual property can be collapsed into an investigation of whether the creation of a right has led to effects which would be noticeable as "monopoly" effects. Actions such as directly restricting supply and raising prices are the usual considerations for determining the presence of monopolies, consistent with Nordhaus's concept of "losses from inefficiencies." This can be observed in comments by Easterbrook who appears to rely on domination of the production market when he argues that patents do not necessarily create


any monopoly:

A patent may create a monopoly - just as an auto manufacturer may own all of the auto production facilities - but property and monopoly usually differ.

Such an approach is narrowly conceived and fails to address wider questions of market power associated with an intellectual property right.

Kitch\textsuperscript{35} relies on this narrow approach in arguing that as competitive pressures are exerted on patent holders, any suggestion of monopoly in its strict sense is false. The argument follows that competition exists in the form of close substitutes. Also, obsolete technology would still be competitive as substantial investments are committed to existing technology and additional investments were required in many cases to shift to new technologies. Hence, near the end of the life of the right, the potential for entry of additional firms would force the patent holder to set prices to a competitive level to discourage entry. Kitch cited Xerox as an example of behaviour where patents did not result in monopoly power, based on evidence that after the introduction of xerography, other forms of document reproduction continued to share the marketplace. This example was criticised by Scherer,\textsuperscript{36} who noted that the allegation of competition was incorrect when the facts were closely examined. Xerox had in fact been engaging in price discrimination in its exercise of market

\begin{footnotesize}
\begin{enumerate}
\end{enumerate}
\end{footnotesize}
Confusion exists in the literature as to whether "monopoly" should be taken to mean the market-distorting effects intended to cure the market failure or in its strict economic sense. Rather than recognising the semantic nature of the argument, authors, particularly those adopting a conventional economics view, suggest that the existence of competitors precludes a finding a monopoly. Chicago writers such as Dam\textsuperscript{38} have noted that unlike monopolies "economic rents" represent an advantage relative to competitors which results in a higher return on investment. Economic rents can be justified in a policy sense as an element of the incentive for invention, and the efficiency question, so far as neo-classical economics is concerned, is whether the existence of rents leads to socially wasteful "rent seeking".\textsuperscript{39} The difficulty is that arguments such as Dam's, in placing such a heavy emphasis on efficiency often marginalise the issue of rents as a problem of equity or allocation. This marginalises the negative impacts of intellectual property rights by assuming that the

\textsuperscript{37} In terms of machine and supplies cost, clerical labour costs and quality, operating a new Xerox machine easily dominated over existing diffusion transfer or Thermofax copiers. The only serious competitor was Electrofax coated paper processes, but for any monthly volume of 1852 copies or above, Xerox was superior. So, Xerox charged 5 cents per copy in the 2000 copies per month range, and 3.5 cents per copy in higher volumes allowing it to dominate the market at the lower prices. Scherer, F.M. (1986) "Comment on Edmund Kitch" Research in Law and Economics, Vol. 8, p. 51 at pp. 53 - 54.


absence of a monopoly, *strictu sensu*, and equating the existence of competitors with “competition”, implies an absence of other undesirable effects. Inevitably the question of "monopoly" effects as against "rents" or "distortions" is essentially one of definitional semantics and rejecting the existence of undesirable economic effects by relying on neo-classical assumptions and definitions detracts significantly from forming a true understanding of the problem. The real issue is not one of a technical monopoly, but rather one of market power.

Scherer's observations above indicate the potential for welfare problems from monopoly which are not captured within a simple demand and supply analysis. Typical welfare losses that can arise are:

- losses due to underutilisation;
- losses due to wasteful/duplicative research; and
- losses due to restriction on invention by acquiring and not using rights.

Underutilisation of the new technology is where consumers willing to purchase the good at its marginal cost of production do not purchase the good, due to the price being raised above that level. The problem is not limited to patent: such a cost has been hypothesised where copyright protection is imposed or increased in scope. For example, Novos and Waldman model copyright protection without price discrimination as to

---

40 Hirschleifer, J. and Riley, J.G. (1979) "The Analytics of Uncertainty and Information -
Chapter Two - Information as a Commodity

quality on the basis that consumers do not vary in their valuations on the quality of the good. In an analytical model this leads them to conclude that copyright protection does result in an underutilisation social welfare loss.41 Where intellectual property rights are stronger, there can be a stimulus to socially wasteful research and development spending to develop technology which is inferior to that protected yet superior to current processes.42

2.5 Under-reward and over-reward

As well as downplaying the significance of monopoly and its direct effects, the "reward for invention" literature appears to marginalise two further issues. One is that the literature assumes an extreme form of competition with costless information flows. This highly competitive market under-rewards invention since competitors will too quickly have imitating products to market at little cost. It has been suggested, however, that natural imitation lags do exist and are sufficient to provide a reward to stimulate invention. Competition itself can be seen as an impetus for invention. The other issue is that intellectual property rights can over-reward invention since it the legal privilege provides the opportunity for anti-competitive behaviour. These issues are discussed below.


Neo-classical economics assesses underinvestment against the benchmark of Pareto optimality. Although Pareto optimality can be a useful tool for the identification of potential problems, it is not useful, at least in terms of intellectual property, for determining their solutions. The underinvestment model relies on an assumption of perfect competition in the market, of numerous buyers and sellers, of homogeneous products, easy entry and perfect knowledge. The assumptions can be explained as arguing that:

- all information is interchangeable;
- all information is communicable;
- all agents are competent in the reception and understanding of information;
- information transfer is costless;
- information transfer is simultaneous (market lead-time provides no advantage);
- no one requires other resources with which the given piece of information must be combined; and
- if there is a requirement of combining pieces of information, the sources of complementarity and potential gain are known.

These assumptions all relate to the process and transmission of information in the context of the nature of invention. They drive the conclusion that, in the absence of intellectual property protection, barriers
to entry into the market are so weak that they offer little assistance to aid appropriability. Sometimes despite the restrictions placed upon it by intellectual property protection, the market is still perceived to fail. Where market power is perceived to be threatened by some exacerbation of the appropriability problem - such as fears of widespread "piracy" across the internet - there follows a perceived need for broadening the scope of the property rights or otherwise strengthening the legal institutions.

This theme within the literature is of under-reward for invention, yet much of this underinvestment literature assumes that within current market institutions little incentive exists to invent except by returns through the exercise of market power contained in intellectual property rights. Indeed, very little is said of benefits otherwise than by royalties.

The following statement by Nelson\textsuperscript{43} is axiomatic to this literature:

Innovation takes R&D resources and involves risks. To motivate the effort under private enterprise requires that the innovator (or those who finance him) can make a profit. Under the ground rules of private enterprise, the only way that this can be done is by having at least a temporary period where the innovating firm and not its competitors can produce the new product or use the new process, or receive compensation from other firms that do.

It has been suggested both conceptually and empirically, however, that competition is one of the most influential factors stimulating research

Indeed, several authors have noted that technological improvements might be just as likely to occur if many manufacturers are competing in the marketplace and that competition may operate as a more effective stimulant to invention than the prospect of monopolistic reward. Scherer, for example, argued that the presence of oligopolistic industry structures, natural imitation lags and the advantages of competitive product leadership, characteristics widespread in the information economy suggest that investment in research and development is wholly rational in the absence of intellectual property protection. This theme has been echoed by Braunstein, Baumol and Mansfield.

Intellectual property may also serve to over-reward invention (and impede later inventors) as it is a means to engage in anti-competitive practices. Examples would include:

- acquisition of related patents so as to control a particular industry or industrial process;

---

• applying financial pressure to smaller firms through litigation;

• accumulation of patents through time so as to perpetuate exclusivity after the expiry of original patents\textsuperscript{49}; and

• application of restrictive licensing conditions\textsuperscript{50}.

These factors are of concern given there is a growing dominance of corporate ownership of intellectual properties and anti-competitive behaviour through exploitation of the rights vested in the ownership of intellectual property\textsuperscript{51}. The economic literature explains this behaviour in terms of cost raising strategies, switching costs, and product differentiation.

Strategies designed to raise rivals' costs have a number of advantages in enhancing market power. Rival's costs may be raised through several means, by:

• raising input prices;

• imposing switching costs;

• increasing expenditure on research and development;

• increasing advertising or promotional expenses;


imposing a "regulatory parameter" – such as intellectual property protection or de facto or de jure standards

Salop and Scheffman\textsuperscript{52} found that cost-raising strategies may result in a decrease in price if the firm seeking to raise rival’s costs has market power in the output market. This result runs counter to those who insist that market power is only reflected in extraction of rents through increase of prices. They found that if the price falls as a result of the action of the dominant firm, the output of the fringe firm falls. Further to this, Salop and Scheffman found that vertical integration can be anti-competitive even where the integration does not monopolise the relevant factor market. A cost-raising strategy linked to product differentiation is that of imposition of switching costs through risk-averse consumption. Time and effort are consumed in learning how to use products. A consumer will usually, therefore, demand that the features of any new product be similar to those features in which the consumer’s learning effort has been expended. For example, features dealing with the interface between the user and a computer program will be valued not only on actual and perceived need, but also on associated costs of obtaining the necessary knowledge to operate the program. These associated costs are switching costs.

Switching costs place fringe firms and new entrants at a cost disadvantage, since to compete they would need to supply additional functionality or

usability to overcome the heavy weight given by the users to interface factors. Switching costs are particularly effective barriers to entry where the extent of vertical integration is asymmetric to the rest of the industry. Established firms with little need to adapt technology from others have a clear cost-incentive to broaden the scope of protection. The implementation of standards, both de facto and de jure, therefore, are a key cost-raising strategy.

Allied to switching costs is the concept of product differentiation. One common method for product differentiation is to direct attention towards the features of the artefact. An equally powerful method is to cultivate a favourable brand image among customers and distributors. The resulting brand loyalty can create a dominant position by raising entry costs for rivals who will have to overcome such loyalty to compete. Brand loyalty may be increased through forward integration (such as buying into the distribution networks), post-sales service, and training of purchasing agents. Incentives exist against ensuring perfection in a product where imperfections allow for a demonstration of support services and a deepening of the personal relationship with the consumer – potentially increasing brand loyalty.

2.6 Summary

This chapter has critically outlined the conventional view of the economic justifications for intellectual property protection. The policy
recommendation – an imposition of a commodity form to information through intellectual property protection – is open to criticism. The dominant perspective is that of a production chain model of invention which generates information as its output.\textsuperscript{53} Intellectual property aims to promote investment in invention through restrictions on the use of the information by others.

Conventional economic analysis assumes that a direct relationship exists between the scope of the monopoly privilege afforded, returns to investors and the level of investment in research and development. Intellectual property is seen as an appropriate intervention to address a market failure which arises from lack of appropriability of information in the hands of users. It is important to note that the perceived appropriability problem is derived from the work of Arrow in dealing with information in a pure sense. Market failure is (allegedly) solved by affording a monopoly privilege: a right to extract rents over the use of information. The argument surrounding the scope of the right is centred on the pursuit of “socially optimal output” in terms of “efficiency”.

A narrow conception of “monopoly” marginalises the effect of welfare losses resulting from underutilisation and wasteful research. The focus on the intellectual property right as the source of return on invention also marginalises other forms of reward and the potential for over-reward through anti-competitive exploitation of intellectual property rights.

\textsuperscript{53} Chapter Four develops an alternative model of innovation (invention) where
The following chapter will outline an alternative and, in the author’s opinion, preferable perspective, based in information economics. The alternative perspective is that information, in the context of intellectual property regulation, is better conceived as a resource. This information perspective is drawn upon to develop the model of invention (technological innovation) presented in Chapter Four, based on the assumption that the information or knowledge underlying the creation of a product is not readily appropriable. Such an approach has profound consequences for the process of innovation and role of intellectual property.
Chapter Three
Information as a Resource

3.1 Introduction

The previous chapter discussed the conventional view of the economics of technological development, which was based on the notion of information as a commodity. This chapter explores the alternative assessment of information – as a resource. Viewing information as a resource facilitates viewing the process of “innovation” as cumulative and dynamic. The main distinction between the resource and commodity views is that a resource has value in use, whilst a commodity has a value in exchange. Information as a resource is better suited than information as a commodity to an analysis of intellectual property regulation, as the aim of intellectual property is to ensure that information is disseminated throughout society. When information is viewed as a commodity, the issue of dissemination of information becomes secondary to a concept of information being traded. This means that considerations of information as an input are effectively disregarded. When viewed as a resource, the role of information in facilitating technological progress is a principal consideration.

54 As stated in Chapter Two, the term “innovation” is used to describe the process of technological progress in abstract. The term is also used to emphasise the incremental nature of the process, particularly drawing upon existing resources, in contrast to the concept of “invention” which suggests an isolated event of inspiration.
In Chapter Two, intellectual property was justified in terms of providing “inventors” with financial incentives to invest in the creation of information products. This incentive to “invent”, however, is not the pivotal focus of affording intellectual property rights within English-derived legal systems. Intellectual property protection’s primary aim has been to maximise the welfare of the community by ensuring “the end” of diffusion of information through “the means” of providing incentives for innovation.56 A vast quantity of legislative and juridical comment, as well as other literature – by far the majority, has commented that the means through which this aim is to be achieved is by weighing the advantages gained by the granting of incentives against the disadvantages flowing to the public from the recognition of property rights.57.

This chapter adopts an information economic perspective to examine the nature of information as a resource for technological innovation. Neoclassical economists have been criticised for failing to account for the role of technology in the economy, and in the dynamics of technological change.58 Recent literature suggests that technology is best viewed as

Chapter Three – Information as a Resource

information as it is composed of information resources of various types.\(^{59}\)

One purpose of this chapter is to outline the properties of information\(^{60}\) from the perspective of information economics, in contrast to the orthodox economics presented in the previous chapter. Another is to discuss the implications for the market of an information perspective, with particular focus on network externalities. Both provide a framework for the development of a model of technological innovation in Chapter Four, and its application to computer programs in Chapter Five.

3.2 Information properties

This section discusses some of the properties of information from an information perspective. This provides a useful contrast with the assumed properties of information from an orthodox perspective, as outlined in Chapter Two. Under the orthodox perspective, information was treated as a form of output from a process of “invention”, and that lack of appropriability suggests that information should be made exclusive by imposition of property rights. The process of information creation was itself seen as an isolated act of inspiration, research and/or development. Under the information perspective, information is seen as a capital resource. Indivisibilities suggest that information should be


\(^{60}\) It is not possible, however, to define information in a precise manner due to the interdisciplinary nature of the literature, much of which is be based in sociology, technical science and psychology: Braman, S. (1989) “Defining Information: An Approach for Policymakers” Telecommunications Policy, Vol. 13, pp. 233-242.
shared to maximise social benefits. The process of information creation is “innovation” which suggests a process which is interactive between firms and based on imitative competition.

3.2.1 Information is capital
A better conception of information is as a capital resource, since it is used to facilitate production. An assumption of information economics is the rejection of the concept of information as a static commodity (an output) and instead adopting the concept of information as dynamic resource (an input). One observation made by Arrow, and largely ignored by later neo-classical authors was that information is not only the product of innovative activity, it is also an input to innovation and prior information is required for the creation of subsequent information. Information may be said to be synthesised and innovated rather than created, building on sources already in the public domain. Technological progress, therefore, must necessarily build on a foundation provided by earlier inventors. Ricketson noted that future creative activity depends upon the public domain and that “copying,

64 Ricketson, S. (1992) New Wine into Old Bottles: Technological Change and
reinterpretation and redefinition” of other works is integral to maintaining a healthy public domain.

3.2.2 Information is shared

The orthodox perspective assumed that information needed to be privately appropriable and thereby made excludable. The information perspective assumes that by its very nature, information’s value to society accrued by common access. The cost of information is independent of the scale on which it is used. In an information intensive economy, information creates pervasive economies of scale. From this proposition Arrow has argued that individuals have clear incentives to organise for effective acquisition and communication of information:

Specialisation in information gathering is one instance, in my view the most important instance, of the economic benefits of organisation. The basic gain in all such cases is that a group working together can produce more in total than the sum of their products working individually . . . Of all the forms of division of labor, the division of information gathering is perhaps the most fundamental.

The theme of the benefits of division of labour is recurring in the context of information. This can be tied into network externality effects generated by the phenomenon of information. Knowledge can be

---


67 See also Leijonhufvud, A. (1989) “Information Costs and the Division of Labour”
internally generated through processes of research, development, design and engineering. It can also be obtained through learning by doing, the acquisition of organisation routines, or from the environment.

As an example of observed sharing behaviour, Rosegger\(^68\) noted an apparent contradiction in behaviour when examining acquisition of knowledge from other firms. Neo-classical theory suggested that both firms and individuals derive returns from proprietary knowledge, hence they should have little incentive to share such knowledge, particularly competitors. Instead, what Rosegger observed was a rapid growth of bilateral, cooperative arrangements and the generation of institutionalised informational exchange on a widening range of topics.

A major element of the incentive to co-operate within information transfer is the need to bring complementary knowledge to bear on the solution of common problems. This model explains why co-operation can be a beneficial strategy for the solution of problems and the development of new technologies. For example, d'Aspremont and Jacquemin\(^69\) suggested that appropriability problems can be mitigated where firms are able to internalise the research and development


spillovers from other firms. Expenditure on research and development is necessary to appropriate the spillovers from other firms and to keep informed about the technological advances of competitors.

The attitude of some policy makers, however, is that valuable information, like valuable physical commodities, should be “protected”, thereby to facilitate its trade. Such thinking is flawed as it fails to appreciate the role of information exchange within the economy. Intellectual property legislation manifests the concern with restricting, privatising and trading information rather than sharing information. Short term commercial incentives to appropriate intellectual property rights of the results of co-operation conflicts with its goals. Short term self-interest at the commercial level can destroy the intended, longer-term co-operation.70 Macdonald and Mandeville71 found that firms in high technology industries engage in extensive second sourcing and cross-licensing as a means to pervert the patent system and that if the patent system were to work as intended, such firms would face significant retardation of information exchange with disastrous effects for innovation and competitiveness.

3.2.3 Innovation is imitation
An important perception within the conventional model is that

innovation is a single, isolated process which takes place within the “black box” of the firm.\textsuperscript{72} Competing firms are seen as attempting to “copy” or “imitate” the original innovation. The policy implication is that the first innovator should be rewarded while its competitors are punished for “appropriating” the former’s investment. Within an information economics perspective, however, information grows synergistically and exponentially in an environment of shared intellectual capital.\textsuperscript{73}

Recently Scotchmer\textsuperscript{74} and colleagues have pursued a theme which is relevant to the conception of knowledge as a resource for innovation. This is the adoption of a perspective which emphasises the cumulative nature of product research development.\textsuperscript{75} A premise in Scotchmer’s work is that firms other than the first innovator produce more creative and diverse second generation products. Without such a premise, the natural solution would be to protect the original innovator so broadly that licensing is required from all second generation innovators.


\textsuperscript{73} Masuda, Y. (1980) \textit{The Information Society as Post-Industrial Society} (Institute for the Information Society: Tokyo).


Scotchmer argues against such an approach. A second innovator who cannot market the next generation product without a license has a very weak bargaining position. Outside firms will either be inefficiently excluded from producing the particular products, or by licensing will be overrewarding the original innovator. According to Scotchmer, when both first and second generation products are developed, the division of profit between the two innovators depends on the breadth of protection. Narrowing the scope of protection is one solution to reward second generation inventors. The challenge is to reward early innovators fully for the technological foundation they provide to later innovators, but to reward later innovators adequately for their improvements and new products as well.

Mandeville argued that within an information-theoretic perspective, any sharp distinction between innovation and imitation – pervasive within the conventional economics of innovation literature – fades away. He rejected a model of one way diffusion of information from an innovating firm to other firms:

The conventional perspective basically seems to assume that technological information is completely embodied in the hardware that emerges from the self-contained, innovating firm. Users and adopters contribute nothing and thus are not innovating. The innovation process stops at the factory gate.

---


Chapter Three - Information as a Resource

Instead, technological progress is evolutionary, social and interactive, a "messy" process involving many participants. This agrees with Schumpeter, who argued that an evolutionary process of technological advance involved a combination of innovation and imitation.

3.3 Information and markets

This section examines the nature of the market in the context of rich information flows. Under an information economic perspective, the market can be characterised by assumptions of bounded rationality and interdependence among economic agents which support strong externalities. One theme in the orthodox literature discussed in the previous chapter was the lack of opportunity to generate returns commensurate to the benefits obtained by users from innovation. This was based on an assumption that information (the product of "invention") was highly appropriable and that a competitive market would drive returns on innovation to zero. The assumptions of the commodity model, based in perfect competition in the market: of numerous buyers and sellers, homogeneous products, easy entry and perfect knowledge, all drive a perception that the market affords poor

78 Ibidem p. 55 (Figure 4.3).
protection to innovators. Information economics provides a different perspective and implications for the market. From an information perspective *homo economicus* is not endowed with perfect knowledge nor necessarily equipped with the means to receive information and correctly process and act upon it. Part of this relies on the nature of tacit and codified knowledge discussed in the following chapter. Information is no longer a simple and objective concept (like a commodity) but adopts qualities contingent on individual and organisation capabilities (like a resource). The assumed non-rivalrous and non-excludable qualities attributable to information are not appropriate in this context. The implications for a market rich in information flows will include:

- lead time;
- bounded rationality; and
- network externalities;

each of which is discussed below.

### 3.3.1 Lead time

Market lead time is perceived to be a greater advantage under an information economic perspective than under a commodity-based perspective. The reason for this added advantage comes from some of Arrow’s later work. Arrow\textsuperscript{81} noted that transmission of information forms only part of communication information.

\textsuperscript{81} Arrow, K.J. “Economic Development: The Present State of the Art” Honolulu, East-
Information exchange is costly not so much because it is hard to transmit but because it is difficult to receive.

So, despite the creation and adoption of information technology to expedite transmission of information, it does not necessarily follow that the costs of communication are decreasing. The ability to interpret and make use of the information must be considered in any depiction of the market for information technology products.

A significant barrier to entry emanates from lead times in production and distribution. First mover advantage results from lags between release of the artefact and the entry of competitors. Such lags can originate from the time which must be taken to:

- recognise market success in an product;
- gain the requisite tacit knowledge and/or reverse engineer the product;
- procure the use of effective marketing and distribution channels;
- secure the availability of complementary information and complementary goods; and
- secure suppliers of raw materials and capital equipment.

Based on these factors, a product will have a greater market lead time advantage the greater the degree of knowledge contained within that

---

West Center 1975 p.18

product and the greater the amount of knowledge required to make use of that product.\(^{84}\) For this reason, in many areas of intellectual endeavour, market lead time advantage can be consciously exploited.

### 3.3.2 Bounded rationality

The more "tacit" the knowledge, the more difficult it is to articulate and communicate.\(^{85}\) Difficulties in communicating knowledge give rise to informational asymmetries.\(^{86}\) Information asymmetries, in turn, mean that rather than people being rational actors making decisions based on perfect information, people try to act with only consideration of the pertinent knowledge at hand available at reasonable cost.\(^{87}\) Industrial and institutional economists, among others, refer to this as "bounded rationality".\(^{88}\) As information must suffer from asymmetries, objectives will pertain to satisficing rather than maximising. Under such a view, economic change occurs under forces of variation, selection and retention.\(^{89}\) Within this framework, variation places emphasis upon

---

\(^{83}\) See Chapter Four

\(^{84}\) This basis for lead time is essential in the model of innovation developed in Chapter Four.


\(^{86}\) Freeman, C. (1982) *The Economics of Industrial Innovation* (Pinter: London)


information and competency asymmetries between firms.\textsuperscript{90} This is in contrast with neo-classical assumptions of agent competence given all relevant information. Variation examines matter such as Heiner's\textsuperscript{91} "Competence-Difficulty" gap: that there is a division between any problem and an agent's competence. For example, limited learning will impair decision making where unexpected yet relevant information becomes available as agents will tend instead to adopt rigid and simplified decision rules thereby relying on circumscribed information. Firms survive by repetition or imitation of successful routines.\textsuperscript{92} In most cases existing routines are slow to adapt and where firms are unable to change quickly enough to meet the challenges of their environment, selection occurs. This process of selection implies that there is a persistent state of disequilibrium within any given industry.

Under the assumption of bounded rationality, the growth of an installed base of technology provides a signal to the market. This signal, in turn, acts to influence search activities and the costs necessary to gather information related to the performance of products. In addition to the search activities, such growth should induce further adoption externalities in respect of costs of retraining, maintenance, and


complementary inputs. Compatibility with the existing installed base of technology and complementarity with other innovations being introduced will often influence adoption choice. This then gives rise to network externalities which can become instrumental in strengthening the irreversibility of the installed base. Pages A critical mass of consensus will define the trajectory of technological development. The outcome of the competition of rivalrous technologies is influenced by size of the existing stock of each of the technology products competing on the market place. Such phenomena also includes what have been described as "bandwagon" effects. This implies that signalling and leadership are important mechanisms for the promotion of, and development within information rich and rapidly changing industries. All these factors are examples of network externalities, discussed below.

3.3.3 Network externalities
External effects are seen as important in may recent studies of technological change, and in the processes of innovation, and diffusion. An "externality" arises where the action of an individual affects other individuals, so that the social costs and benefits do not agree with the individual costs and benefits. Negative externalities such as pollution

Chapter Three – Information as a Resource

arise since individuals do not bear the direct costs of pollution and therefore have adverse incentives to pollute if it derives economic gains to them personally. Direct externalities, whether positive (beneficial) or negative (harmful) “lead the Invisible Hand astray”\(^96\). At the most elementary level, it is clear that telecommunications networks provide positive strategic externalities beyond initially quantifiable costs and benefits. Gombani\(^97\) describes information networks as being at the forefront of the transition process from the age of machines to the age of information. Pervasity of information networks is a significant feature of what he states is the “discontinuity” period society is currently experiencing, marked by continuous technological advances. Network economics is an attempt to characterise the:\(^98\)

- static and dynamic features of a system of interaction among agents where a variety of signals, besides prices, matter and where hysteresis plays a major role;
- pervasive role of technical, pecuniary, adoption and consumption externalities in the introduction of technological change;
- strategic attitude of firms to internalise external factors through measures to reduce uncertainty and increasing command of

---


networks are the result of interdependence between agents which is not expressed by prices, but by a broader array of signals and quantitative and qualitative information. The typical example is of a telephone network. The value to an individual of being connected to the network is related in non-linear fashion to the number of other people connected to the network. Markets perform poorly in the presence of strong externalities since prices do not convey all relevant information so decision making may need to be co-ordinated through other signalling processes, such as co-operation between firms or government intervention.

Despite semantic arguments to the contrary,99 network externalities are accepted as significant in many important industries where there are no physical networks.100 Any technology that requires specific training is subject to network externalities since the training is more valuable if the associated technology is more widely adopted. This forms part of "adoption externalities". Adoption externalities are characterised by the:

- interdependence between potential users and actual adopters of a new technology;

- role in the diffusion process of adoption externalities engendered


by the adoption behaviour; and

• creation and maintenance of skills through use of the technology.

The first characteristic, interdependence of one agent's utility in consumption with the behaviour of other agents in the economy in respect of their consumption choices, can be more simply phrased: people's preferences are interrelated with what other people will want to buy.¹⁰¹ The dynamics of adoption externalities affect the pricing and performance of new products, and have serious effects on the demand for information technology artefacts. Demand and diffusion effects of a new artefact are closely correlated as the benefit which a consumer derives from the use of a good is often an increasing function of the number of other consumers purchasing compatible items. This is particularly so where complementary goods are required to use the product effectively or to achieve significantly increased utility for the consumer. Many examples of network effects entail indirect externalities associated with the provision of a durable goods and a complementary good or service. In these cases the externalities arise when the amount and variety of complementary goods available increase with the number of durable good units sold. For instance, computers and programs must be used together to produce computing services and the greater the sales of hardware, the more surplus the consumer is likely to enjoy in the software market due to increased entry. An interesting point is that very strong network

¹⁰¹ Here, network economics diverges widely from neo-classical economics, as neo-
externalities may suggest a demand and supply curves behaving similarly to a Veblen distribution, as people take the increase in price to signal an increase in functionality, usability or complementarity with other products.

Both external\textsuperscript{102} and internal\textsuperscript{103} information networks facilitate the exploitation of information by organisations. Once it is accepted that innovation is a matter of gathering, assembling and transforming information into knowledge, information networks can be seen to have a role in directing the process of innovation.

An example of economies of scope is where a product may have functions or features such that a combination of such functions is more valuable to the consumer than each of the functions independently. Although economies of scope can be achieved by product variety\textsuperscript{104} the interdependence implied by the generation of economies between these products should also result in network economies of inputs to production and also feedback effects.\textsuperscript{105} Where there is an existing market lead time classical analysis assumes consumer tastes, technology and market structures as given.

\begin{enumerate}
\item Lamberton, D.M. (1988) "Communication in Economic Processes" in More, E. and Lewis,
advantage, plus network externalities, depending on the nature of the product, the firm may be able to consciously generate and exploit economies of scope by including additional functionality or usability to deliberately setting up a demand for complementary products or services.

The concept of network externalities discussed above is an important feature of informationally rich markets which contrasts with the commodity based perspective of orthodox economics. Network externalities can be the source of enormous benefits to consumers but may also provide the foundation for significant market power by producers. An information perspective suggests that the realisation of network externalities – enhancement of information flows – is critically important for the design of intellectual property rights. The realisation of network externalities in the area computer programs is pursued in Chapter Six.

3.4 Summary

Viewing information as dynamic and technological progress as cumulative alters the perception of which factors will influence the market for information technology artefacts.

Information is a capital resource, the cost of which is independent of the scale on which it is used. One implication is that a pattern of sharing of this intellectual capital is rational. Another is that innovation is an
ongoing, cumulative process of imitation and learning. An information perspective also holds distinct implications for the market. Lead times in production and distribution can be significant the greater the degree of information embodied in a product and the greater the degree of knowledge required to make use of it. This suggested that information was to a degree and for a time excludable. In contrast to the assumptions of perfect competition and costless information flows, information economics suggests that the economy is likely to exhibit significant lead times and make investment in innovation rational in the absence of intellectual property protection. Given limited information processing abilities and limited available information economic actors behave in a "bounded rationality" which suggests innovation based on trajectories supported by product leadership and bandwagon effects. A market characterised by network externalities will not exhibit the same behaviour as commodity based markets and in particular provides scope for significant exploitation of market power.

This chapter has explored the perspective of information as a resource and its consequences for economic thinking. It is the basis for the development of the innovation model in Chapter Four which is applied to computer programs in Chapter Five.
Chapter Four
An Informational Model of Technological Innovation

4.1 Introduction

Chapters Two and Three have outlined and contrasted the orthodox and information economics perspectives on the question of technological innovation. The information perspective is adopted in this thesis as appropriate to the question of intellectual property in general, and to computer programs in particular. One objective of this thesis is to use information economics to develop a model of innovation within the software industry and to use this model as a benchmark to assess copyright policy in this context.

This chapter develops an informational model of technological innovation. The model is developed from two conceptual distinctions. Both distinctions provide a counterpoint to the conventional view that technology is highly appropriable. The first is a division between tacit and codified knowledge drawn from Polyani’s theory of tacit knowing and Mandeville’s model of codified information and technological change. The process of transforming tacit to codified

---

information is part of the process of innovation and it is costly and excludable. The second distinction is drawn from Monk’s distinction between “technology” and “technological artefacts”; the means of production and the product. This distinction rejects the assumption – implicit in much of the conventional economic literature – that distribution of the applied technology necessarily discloses the knowledge underlying its creation. The economic implications of the model are discussed and the model integrated with fundamental concepts of intellectual property doctrine, with particular focus on the idea-expression dichotomy within copyright.

4.2 Tacit and codified knowledge

The idea that awareness follows a characteristic structure, whereby particular aspects of consciousness are known subsidiarily and depend on the particular conscious focus, comes from Polyani, who argued that:107

All knowledge falls into one of these two classes: it is either tacit or rooted in tacit information. The idea of strictly explicit knowledge is indeed self-contradictory; deprived of their tacit coefficients, all spoken words, all formulae, all maps and graphs, are strictly meaningless.

The subsidiary element is known as “tacit knowledge”. Subsidiary elements need not be vague or at the edge of consciousness, they may range from scientifically ordered and coherent facts to unspecified or even unspecifiable - “ineffable” - elements.

This thesis adopts a narrower definition of which is “tacit” such that “tacit knowledge” represents intangible information only. Thus “tacit knowledge” as used in this thesis is essentially identical to the concept of “uncodified information” used by Mandeville, and is consistent with a range of parallel concepts adopted by innovation theorists as set out below:

Table 4-1: Parallel Concepts to Tacit and Codified Knowledge

<table>
<thead>
<tr>
<th>Uncodified Information</th>
<th>Codified Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schumpeter’s “invention”</td>
<td>Schumpeter’s “innovation”</td>
</tr>
<tr>
<td>Nelson’s “logy”</td>
<td>Nelson’s “technique”</td>
</tr>
<tr>
<td>Mansfield’s “unembodied” technology</td>
<td>Mansfield’s “embodied” technology</td>
</tr>
<tr>
<td>Macdonald’s “innovative process”</td>
<td>Macdonald’s “innovation process”</td>
</tr>
<tr>
<td>Abernathy’s “fluid” state of technology</td>
<td>Abernathy’s “specific” state of technology</td>
</tr>
<tr>
<td>Teece’s “leading edge” technology</td>
<td>Teece’s “state of the art” technology</td>
</tr>
<tr>
<td>Jensen and Macdonald’s “technology”</td>
<td>Jensen and Macdonald’s “technique”</td>
</tr>
<tr>
<td>Leonard-Barton and Roger’s “horizontal” diffusion model</td>
<td>Leonard-Barton and Roger’s “classical” diffusion model</td>
</tr>
</tbody>
</table>


The personal nature of tacit knowledge gives rise to an important characteristic: it is not in a state which is readily communicable to others. This is crucial, as information cannot diffuse through society without communication. To aid in its communication and to implement the knowledge in production, tacit information is codified. The manufacture of an information technology product in turn relies on codified information such as formulae, laboratory procedures and blueprints.110 Codified knowledge is in a form which is intended to permit its communication. In a codified form knowledge becomes disembodied from individuals and would be described by some writers as “information” rather than knowledge. Knowledge, therefore, can be viewed as a “matrix” of information which has been contextualised and digested either consciously or subconsciously:111

Knowledge is not a pile of homogenous material, but a complex structure of heterogenous thoughts, each available at zero marginal cost but usable only together with resources available only at positive, and often very high cost.

As an example, a bread recipe listing ingredients and approximate cooking times represents codified knowledge. Underlying this codified knowledge will be a body of tacit knowledge which includes techniques of sifting,


kneading, yeast preparation (depending on the variety) and judgment about precise cooking times based on appearance and other factors.

The process of turning tacit knowledge into codified knowledge (information) by giving it form is described by Polanyi as “articulation”. Articulation is typically achieved through symbolic representations in language, mathematics, graphs and pictures. The “text” produced through this process of articulation specifies not only the particulars of experience but also the pattern of their integration: a conceptual framework referenced by the environment. This suggests that codified knowledge is derived from a variety of formal and informal information channels and will rarely be limited to processes internal to a particular firm. Successful innovators, for example, are those who are sensitive to feedback mechanisms particularly involving understanding of user needs.112

The ability to transmit codified knowledge as information does not necessarily ensure understanding by the person to whom it is being transmitted. For example, possession of a recipe book does not imply mastery of the relevant skills to bake bread of high quality or at all. To understand the recipe book, one would need in turn to understand the terminology, the use of kitchen appliances and in many cases the process of conversion between metric and imperial scales. As a preliminary factor

the possessor of the book needs also to understand the language in which the book is expressed (another form of tacit knowledge). This is a useful example, as it highlights one further aspect. Commodity-based arguments presume that knowledge is perfectly and costlessly communicable, yet particular skills are required to receive and make sense of information before it becomes knowledge. The ability to use a technology is only partly explained by the existence of codified sources. Practice, and feedback through criticism, are necessary for the learning and retention of many skills. Extending the cooking example, learning a baking process is significantly enhanced when a more experienced chef (mother) watches, explains, directs and corrects. This relates to Von Hippel’s observations on learning by doing and learning by using.\textsuperscript{113} It should also be noted that Monk\textsuperscript{114} used the additional terms “blueprint knowledge” to describe semi-formal and empirical information resources relating to product design, and “theoretical knowledge” to representing more formal and analytical information resources. These terms contrast against Monk’s characterisation of “tacit knowledge” as essentially informal and the product of learning by doing.

This distinction between tacit and codified knowledge challenges the convention assumptions of technology exhibiting “public good” characteristics, in the sense of expensive production and inexpensive

reproduction. Tacit knowledge is not a public good since it requires a learning process on the part of the recipient which can be costly and time consuming. Information needs to flow through capable individuals. Lamberton argues that modelling of innovation needs to recognise capability as an important component of organisational capital. Significant absorption costs qualify the public good assumption conventional theory attaches to technologies.

The importance of the distinction between tacit and codified knowledge is that is challenges the orthodox assumption that information is highly appropriable as discussed in the previous chapters. The development of a model of information technology “artefacts” in the following section seeks to pursue a conception of technological development which involves products whose underlying technologies are not appropriable and do provide incentives to innovate without strong intellectual property protection.

4.3 A model of IT innovation

Monk has analysed technological change and the role of information


resources. He suggested that analyses of the economics of innovation have been focussed on the characteristics of its producers and the nature of the market for its products, rather than the technology itself.

Neo-classical economists have been criticised for failing to account for the role of technology in the economy and in the dynamics of technological change. One of Monk’s significant contributions was that a distinction can be drawn between “technology” – the knowledge or ability required to enable economic production – and “artefacts”, which are the physical embodiments of information. Technology is composed of coherently related sets of information while technological artefacts are physical products which result from application of the technology to production. These artefacts are labelled “information technology artefacts” (“IT artefacts”) in subsequent discussion. The importance of Monk’s distinction is that:

---


122 “Technology” must have some present or potential value in use in production. Information which has no potential use value in production cannot be considered as part of “technology”.

It allows the existence of [information] resources to be considered separately from the form in which they are embodied and from the forms in which they are implemented in economic production. Confusion between the existence, embodiment and implementation of technological information resources may account for some of the persistent difficulties and contradictions found in the literature.

As an example, the science of engineering is a type of technology. A machine, bridge, or chemical compound would be an artefact. More appropriately for this thesis, computer science constitutes an information technology, while software is an information technology artefact. A graphical representation of the concept of IT artefacts outlined in this section is found in Figure 4-1.

IT artefacts will have significantly different economic characteristics from their underlying technologies. The significant characteristic for the purposes of the model in this thesis is that the knowledge underlying the creation of IT artefacts is not readily discernible to the person using the artefact. This is an important assumption since it implies that the artefact will not suffer from the same degree of appropriability and market failure (in the absence of intellectual property protection) which is assumed by conventional analysis. The conventional view outlined in chapter two assumes that technology is always highly codified in the sense used by Mandeville, and competitors or users can quickly and cheaply appropriate the knowledge underlying an IT artefact.

Figure 4-1 - Information Technology Artefacts

- **Embodies knowledge, but does not transmit knowledge about the means of production of the artefact**
- **Means for production of IT artefact (tacit and codified knowledge)**
- **Value in use of IT artefact appropriable to**
- **INCENTIVE TO INNOVATE**
This thesis rejects the conventional analysis. An IT artefact is generally excludable since it is typically a physical good and its value in use is thereby appropriable from users. The information (or technology) underlying its creation will typically be difficult or impossible to access merely from its use. At this point it is important to distinguish between the technological elements embodied in artefacts and their surface design. For example, a Rolex watch can be "copied" on the basis that the copy is in surface design and behaviour similar to the original. The copier is likely, however, to have appropriated little if any of the craftsmanship of the maker of the original. Even internal mechanisms of the watch might be reproduced, but this can be done without any understanding of the engineering underlying the mechanism or the aesthetics of the artistic composition involved.

An IT artefact will not, under these conditions, display "public good" characteristics. The knowledge underlying production is not merely appropriated by access to the physical product. It follows that a sufficient incentive to innovate can exist provided at least that the reproduction (in the sense of replication) of the IT artefact is restricted.

An information technology model of innovation derived from the above analysis is presented in Figure 4-2. The model can be explained by a focus on critical information flows represented by the arrows and elaborated in

---

126 A theme pursued below in chapter six.

Figure 4-2 - An Informational Model of Technological Innovation

Chapter Four - An Informational Model of Technological Innovation

Users

Tacit Knowledge

Codified Knowledge

IT Artefact X

Users

IT Artefact Y

Users

Firm X

Learning by using, feedback

Firm Y

Articulation

Production

Sale/Licensing

Learning by doing
the following discussion. Innovation is determined by the tacit knowledge accumulated within individuals, including individuals within a particular firm. Tacit knowledge is important to developers in that it represents part of their capital in the form of organisational capability of information handling. Much of this knowledge base is not in a codified form and represents individual know-how or the know-how of several individuals ordered into organisational routines. It is important to note that knowledge must be accumulated within individuals and although firms may have legal personality, they are independently incapable of possessing knowledge. This privately held collection of tacit knowledge represents part of the capacity for innovation.

The tacit knowledge of individuals within a firm will accumulate over time and derive in part from the process of articulating (codifying) the tacit knowledge so as to apply it to production. This suggests a two-way interaction between tacit and codified knowledge inherent in the concept of learning by doing. The firm’s innovative activity must, at least to a degree, draw upon other sources including the “public domain”. Information acquired from the environment – the public domain – will include the information networks existing not only between rival firms but between users of the IT artefacts. The model explicitly links the

informational flows of users to those of the firm. Monk\textsuperscript{128} stated that the tacit knowledge of both developers and users is crucial to the practical evolution and implementation of technology. It is this relationship which most clearly suggests the existence of bounded rationality and network externalities within the process of innovation. The tacit knowledge influencing innovation incorporates important information from the marketplace such as product leadership, complementarities and economies of scope. This suggests a far more untidy and complex interaction with other individuals and firms as well as with existing (typically competing) information artefacts. Adopting a cumulative and interactive approach to innovation implies that technology must develop along a trajectory signposted by selection of successful routines.

\section*{4.4 IT innovation and intellectual property}

Having developed a model of technological innovation in the previous section, the following discussion applies this model to the question of innovation and the role of intellectual property in regulating information flows. As discussed in Chapter Three, the promotion or restriction of information flows within the economy can have profound effects. The model is applied to three major intellectual property regimes: confidential information, patent, and copyright. The model is modified where appropriate to suit the assumptions of the particular regime. Copyright as

\textsuperscript{128} Monk, P. (1989) \textit{Technological Change in the Information Economy} (Pinter Publishers) p. 79.
the major source of protection for computer programs is given some emphasis.

One implication from the model which follows from Monk's work is that although the "absolute" state of technology depends on the currently existing information set, the "effective" state of technology depends on the availability, distribution and allocation of the manifest forms of the knowledge. Consider this illustration: a discovery of cold fusion by technicians working for the Department of Defence would constitute a change in the absolute state of technology yet if this knowledge was kept secret for strategic purposes, there would be no effective change in the state of technology from the perspective of the rest of the world. "Absolute" technological change, therefore, occurs when the set of knowledge and abilities in the economy increases, irrespective of the availability of the technology - its social distribution. An "effective" technological change occurs where there is a change in the availability of information resources throughout the economy. This thesis takes Monk's argument one step further and argues that real technological change is promoted by a broadening of tacit knowledge. This means that policy makers should not only consider the availability of particular products to consumers, but also the distribution through society of the means by which such products are created. Policies should be adopted to ensure the promotion of the articulation of these means in forms which permit their dissemination. This section demonstrates why intellectual property laws can be seen as designed in such a way so as to enhance
4.4.1 Confidential information

In an unregulated environment, information would flow between firms. This environment is illustrated in Figure 4-3. In particular, three type of flows would be expected. One is the exchange of tacit knowledge (industrial know-how) through processes of personal interaction, informal networks, and employee mobility. Another is a learning process where codified knowledge of one firm (blueprints, marketing information etc) is transmitted to another firm and becomes the subject of study. This information might be incorporated directly into the production process of the firm, but is more likely to result in the individuals within the second firm learning new or different means of production. Rosegger\textsuperscript{129} notes that copying of "the letters of learned men" in a growing network of postal services through Europe in the 16th and 17th centuries contributed to the spread of new ideas. Harris\textsuperscript{130} notes the role of espionage in effecting international information transfer during the 18th century. Employees at a follow-on firm may, as a result of access to codified information of the originating firm, deduce alternative solutions to particular applied problems. A third information flow, represented in blue on Figure 4-3, is that of environmental factors where information is obtained from study of the IT artefact and market feedback,


\textsuperscript{130} Harris, J. (1986) "Spies who Sparked the Industrial Revolution" New Scientist (22 page 68
Figure 4-3 - Confidential Information

[Diagram showing the relationship of confidential information between Tacit Knowledge and Codified Knowledge in Firm X and Firm Y, with arrows indicating the transmission of information and personal interaction.]
rather than from sources internal to other firms. In generic terms this may be described as "reverse engineering".

These information flows are, however, regulated at a basic level by the laws governing confidential information. In many jurisdictions this type of legal right is described as "trade secret" protection. In Anglo-Australian law, breach of confidence is protected by a number of distinct juristic bases including property rights\textsuperscript{131}, fiduciary relationships\textsuperscript{132}, and related obligations of good faith. In some instances (such as employer-employee relationships) it will be supported by contractual obligations. It is well established that a wide variety of commercial and industrial information can be subject to protection.\textsuperscript{133} The areas boxed in red on Figure 4-3 represent a relationship of confidentiality. The information flows between firms may be restricted by the use of confidential information laws. The information flows from the general environment are not restricted.

Davidson\textsuperscript{134}, adopting a "good faith" analysis, noted that protection of confidential information, in contrast to patent and copyright, was not intended to encourage dissemination of innovative knowledge through

\begin{footnotesize}
\begin{itemize}
\item \textsuperscript{131} Such as common law rights of property in unpublished works: \textit{Prince Albert v Strange} (1849) 47 ER 1302; \textit{Philip v Pennell} [1907] 2 Ch 577.
\item \textsuperscript{132} Jones, G. (1968) "Unjust Enrichment and the Fiduciary's Duty of Loyalty" \textit{Law Quarterly Review}, Vol. 84, p. 472.
\item \textsuperscript{134} Davidson, D.M. (1983) "Protecting Computer Software: A Comprehensive Analysis"
\end{itemize}
\end{footnotesize}
society through disclosure. Instead, its purpose is to maintain standards of commercial ethics as well as encourage the generation of competitive advantage. This competitive advantage is typically manifest in market lead time, as the process of other firms imitating the IT artefact without access to the secret tacit and codified knowledge will be costly and time consuming.

Reichman135 argued that "trade secrets" law has social value in that the task of reverse engineering unpatented, undisclosed innovation contributes to the innovative community's overall costs of research and development. Reverse engineering provides originators with natural lead time. Only followers who are prepared to defray the costs of study, analysis, and (in many cases) relearning will be able to exploit the cost reductions, technical improvements and potential new applications arising from the reverse engineering process. Reichman stated:136

The technical community as a whole thus benefits from self-help diffusion of unpatentable skills and knowledge, while each competing firm's implementation of the resulting improvements and new applications becomes one of the best means of undermining the goodwill lawfully accruing to the first comer's own trademark.

The legal principles regulating confidential information can be seen, within the information model, as promoting the growth of tacit knowledge in society by the encouragement of learning by imitation. This

---

is because regardless as to whether confidential information has a proprietary or an equitable base, most jurisdictions permit independent creation. In contrast to the conventional model, the incentive to innovate does not arise from monopoly rights, but from competitive pressures\textsuperscript{137} and natural lead time. The presumed secrecy of the tacit and codified knowledge underlying this form of regulation agrees with the assumption that mere possession and use of at least some products ("IT artefacts") is insufficient to understand the technology underlying its production. This requires costly and time consuming analysis and learning. Yet, trade secrets ultimately encourage research and development as there is an incentive to imitate successful products. An example of this can be seen in the research conducted by Coca Cola’s competitors to refine a drink with the same taste. Confidential information model most closely fits the concept of the IT artefact discussed above.

4.4.2 Patent
Patents are a particular form of property created by legislation. In Australia, a patent creates a monopoly over the exploitation of innovations in the sense of a "manner of new manufacture", broadly conceived as technology develops over time.\textsuperscript{138} Subject matter which is

\textsuperscript{136} Ibidem p. 2522.
\textsuperscript{137} Discussed in section 3.3.1.
\textsuperscript{138} National Research Development Corporation v Commissioner of Patents (1959) 102 CLR 252 at 271 per Dixon CJ, Kitto and Windeyer J.
Chapter Four – An Informational Model of Technological Innovation

patentable is also limited by the requirements of novelty and non-obviousness. The patent laws grants exclusive rights to make use of, assign, or license the innovation for the term of the patent, which is limited to a time period specified by domestic legislation – typically 20 years. Patent does provide a limited monopoly and does protect against independent creation but the aim of providing patent protection is that by extensive disclosure, the innovation should become part of the ordinary stock of technology which is available for use by all manufacturers. Silberston, for example, speaks of the importance of the “publication of new technical specifications” facilitating the “rapid spread” of knowledge.

The application of the patent system to the innovation model is presented in Figure 4-4. There are three major information flows identified. The first is the disclosure of codified knowledge by the patent holder (Firm X).

An important element of the patent system is the disclosure of the patented innovation through registration. Firms can make direct use of this information either through licensing from the patent holder or by waiting until the expiry of the patent term. Conventional analysis would suggest that mere disclosure permits rival firms to appropriate the means


Figure 4-4 - Patents

Chapter Four - An Informational Model of Technological Innovation

Firm X

- Tacit Knowledge
- Codified Knowledge
- Disclosure to other firms
- Learning

Firm Y

- Tacit Knowledge
- Codified Knowledge

Patent disclosure (licensing or expiry)

Users

IF Artefact X

IF Artefact Y

Prohibition of reverse engineering

page 72A
of production at little cost or effort. Users and adopters are perceived to be contributing little to the innovation process, or are simply replicating the innovation.\textsuperscript{142} In the graphical representation, the typical perception would be that, by accessing the patent of Firm X, Firm Y is producing a clone of IT Artefact X rather than an improved, specialised, or cheaper IT Artefact Y.

An informational approach would suggest that disclosure does not lead to perfect appropriation. A patent holder only makes available the means of production in a codified form – a great deal of technological know how is, however, in a tacit form. For this reason many patent licences are accompanied by “know-how agreements” where the licensing firm provides consulting staff in order that the patent can be used. Imitation is therefore not appropriation but a transfer and learning process. This represents the second information flow, between tacit and codified knowledge within Firm Y, in blue in Figure 4-4. An imitating firm will require considerable investment on learning as well as direct production costs, which suggests that imitating firms would not be in a superior economic position to originators. This implies, even under a patent system, considerable natural protection for technological innovation.\textsuperscript{143}

The serious impact of a patent system is the restriction on the flow of information from the environment which, as discussed above, would be

the dominant form of innovation in an unregulated environment, or one regulated simply by relationships of confidence. In an informational model the imposition of this legal structure on the process of innovation implicitly assumes that, for some industries and/or innovations, the unregulated solution - reverse engineering - would too slow, costly, or difficult without disclosure from the original innovator(s). This conclusion is supported by the existence of the requirements of novelty and nonobviousness for patent protection to be granted.

In Figure 4-4, the dotted green line represents the restriction on reverse engineering imposed by a patent system. The intention of the patent system is that firms should engage in learning from information licensed by the patent holder or not at all. Macdonald and Mandeville\textsuperscript{144} found, however, that firms in high technology industries engage in extensive second sourcing and cross-licensing as a means to pervert the patent system and that if the patent system were to work as intended, such firms would face significant retardation of information exchange with disastrous effects for innovation and competitiveness.

4.4.3 Copyright

In general terms, copyright provides legal redress to authors where their literary, dramatic, musical or artistic works as well as subject matters other

\textsuperscript{143} Ibidem pp. 93 - 94.

than works\textsuperscript{145} have been unfairly reproduced without their permission. Copyright protection is granted by specific legislation\textsuperscript{146}. The scope of protection depends not only on judicial interpretation of the legislation, but on the kind of work involved. Current copyright law finds infringement in a wide variety of direct and indirect forms of reproduction and adaptation. The term of protection is limited to the life of the author plus 50 years\textsuperscript{147}. No formalities are required for copyright protection as for patent. In order to determine infringement, copyright must be found to subsist in the plaintiff's work and the defendant must, on the balance of probabilities, be proven to have copied a "substantial part" of that work\textsuperscript{148}. Substantial taking is judged by analysing the \textit{quality} of the part taken rather than the \textit{quantity}\textsuperscript{149}. This is because, although a large quantity may have been taken, it may have been relatively unoriginal – the obverse is also true. An investigation of the originality of what is taken is essential in determining the question of substantiality\textsuperscript{150}. For copyright to subsist in a work its reduction to material form must contain an element of the author's intellectual output. Elements not resulting from this effort are not protected.

\textsuperscript{145} Copyright also includes other subject matter such as broadcast, sound recordings, cinematograph films, and published editions.

\textsuperscript{146} In Australia, by the \textit{Copyright Act} 1968 (Cth) (“CA”).

\textsuperscript{147} CA s. 33(2).

\textsuperscript{148} CA s. 14(1)(a).

\textsuperscript{149} \textit{Ladbroke (Football) Ltd v William Hill (Football) Ltd} [1964] 1 WLR 273 at 293 per Lord Pearce.
Chapter Four – An Informational Model of Technological Innovation

Kreiss\textsuperscript{151} argues that the basic purpose of copyright law is to advance learning and knowledge. The laws constitutes an incentive system which encourages the production and dissemination of works, but to achieve the purposes of copyright, such works must be accessible. Under this model copyright constitutes a \textit{quid pro quo} in which commercial rewards flowing to producers is balanced by the increase in knowledge in society, through a learning process stemming from the works' dissemination. This is but one of a number of analyses whose metaphor is a balancing process between exclusivity and accessibility.

Copyright law addresses the balancing process by use of two principal legal tools: the idea-expression dichotomy and the doctrine of fair use. A third mechanism is the expiry of the copyright term, but for technological industries, this may be of little value. The idea-expression dichotomy has been of most significance in decisions relating to IT artefacts and computer software in particular. The dichotomy upholds the principle of access to works in that while ideas contained in copyrighted works are freely available, the particular form of expression of those ideas may not be reproduced or adapted without permission.\textsuperscript{152} A concise statement of

\textsuperscript{150} \textit{Warwick Films v Eisinger} [1969] Ch 508.


the principle was made by Lindley LJ in *Hollinrake v Truswell*:

Copyright, however, does not extend to ideas, or schemes, or systems, or methods; it is confined to their expression: and if their expression is not copied the copyright is not infringed.

While agreeable in abstract, the concept is difficult to apply in practice. Netanel describes it as “notoriously malleable and indeterminate”. A few commentators favouring stronger and broader protection have argued against the application of this principle, and in favour of protection of ideas. This distinction does, however, accord with much of the economic literature surrounding intellectual property. Landes and Posner, for example, examine copyright within a formal economic model of the effects of protection on the production of creative works. The model uses a single variable as a proxy for a range of policy decisions including the degree of similarity triggering infringement, the elements within a work which are protected, and the duration of the property right. The up-front fixed "cost of expression", principally the input of the author in creating the work, is treated as a fixed cost variable.

The effect of an increase in copyright protection is both to increase the

---

153 [1894] 3 Ch 420 at 427.
author's fixed cost of expression and the copiers' marginal cost of reproduction. The cost of expression is increased since a broadening of copyright protection will limit the availability of public domain material from which new works can be created, also, this may force the author to take costly steps to avoid infringement of existing works. As the level of copyright protection increases, through broadening or lengthening the scope of protection, there are two effects. One is that the potential profits of authors will rise, increasing the number of works produced (a movement along the supply curve) the other is that an increase in protection drives up the cost of expression (a shift to the left of the supply curve). Between a negligible amount of protection and the theoretically optimal level, the effect on authors' expected profits will dominate, increasing socially desirable works, but only up to a point. Beyond a critical level of copyright the effect of further expansion of copyright will both drive out copiers and also increasing the cost of expression so as to reduce the number of works created since the public domain will be more limited. Landes and Posner\(^{157}\) state:

Some copyright protection is necessary to generate the incentives to incur the costs of creating easily copied works, but too much protection can raise the costs of creation of subsequent authors to the point where those authors cannot cover them even though they have complete copyright protection for their own originality.

Some significant implications can be drawn from the Landes and Posner model which provide the basis for a framework for analysis of copyright.

\(^{157}\) \textit{Ibidem} p. 335.
Firstly, the "optimal" level of copyright (that which maximises welfare) is demonstrated to be at a level below that which maximises the number of works produced, or provides the maximum return to the producers of the works. This emphasises the importance of striking a balance between the need to provide incentives and the desirability for society's access to works, particularly to technological advances. Landes and Posner\textsuperscript{158} state that copyright protection:

\ldots trades off the costs of limiting access to a work against the benefits of providing incentives to create the work in the first place. Striking the correct balance between access and incentives is the central problem in copyright law.

Secondly, the more the cost of expression increases with respect to the broadening of copyright protection the lower will be the optimal level of copyright protection. Copiers in the sense that Landes and Posner define them are limited to literal copying rather than non-literal copying such as reverse engineering. There would be a separate demand curve for non-literal copies. This suggests that it is socially desirable to differentiate between literal copiers and those using the work to create new works. Protection should not be as extensive against the latter group. This accords with the informational model of innovation used in this chapter.

Finally, Landes and Posner suggest that \textit{ex ante} authors would, behind a veil of ignorance, choose a copyright rule which protected expression but not ideas. In theory, there is a level of copyright which policy setters will

\textsuperscript{158} \textit{Ibidem} p. 326.
seek to attune the law which evenly balances the increase in the cost of creating new works against the increase in revenues by copyright holders of existing works. This would be because they would not know whether they were to be the first or a later generation author.

Copyright as applied to the innovation model is presented in Figure 4-5. Similar to the situation in the applications discussed above, part of the innovation process will involve the articulation of tacit knowledge into a codified form. In broad terms, tacit knowledge can be equated with copyright’s conception of “ideas” and the process of codification as “expression” of these ideas. This process is identified in red on the figure.

The significant difference between this depiction of the innovation process and those of the other intellectual property situations is that the codified knowledge is no longer separable from the IT artefact. For literary, dramatic and artistic works the codified knowledge is no longer a means to an end (means of production) but an end in itself. The artefact no longer has the characteristics of a “knowledge artefact” in the sense used by Monk. Since it is now a joint product (represented in blue) possession of a protected work will permit appropriation by users who will not contribute to the innovation process themselves. Copyright law, principally through the mechanism of the idea-expression dichotomy, promotes socially useful innovation by prohibiting replicative reproduction of the work but permitting further innovations by others based on similar ideas, systems and methods. This information flow is represented in green on Figure 4-5. Importantly, these ideas are not
Chapter Four - An Informational Model of Technological Innovation

Figure 4-5 - Copyright
simply obtained from the work itself, but also involve participants in the marketplace. Innovation, even in artistic endeavours, should progress along particular trajectories based on bounded rationality. This is represented as information flowing through the network of users.

The important policy implication is that maintenance of a public domain of accessible ideas is important for the process of innovation. Yen\textsuperscript{159} notes:

\begin{quote}
All authors borrow from other works. By prohibiting borrowing from copyrighted works, copyright makes it harder for authors to produce new works. Those authors must pay to borrow, or they start from scratch. These obstacles hinder the production of beneficial works, thereby retarding "the progress of Science and useful Arts".
\end{quote}

If copyright is to govern innovation within a particular industry, striking the correct balance, or drawing the most appropriate line between idea and expression, will therefore be major determinant of the economic benefits and costs.

\section*{4.5 Summary}

This chapter developed a model of technological innovation from an information perspective. This model derives from literature within an information economics perspective and is based on two major theoretical constructs:

• a distinction between tacit and codified knowledge; and
• a distinction between knowledge and knowledge artefacts.

Both these distinctions are at odds with orthodox literature which treats information as exhibiting "public good" characteristics. The model of "IT artefacts" as commodities embodying technological knowledge instead assumes that the underlying knowledge is not readily appropriable from possession of that artefact. This suggests that the market failure identified by Arrow will not apply to IT artefacts in the same way it might apply to information in a codified form. Competitors will not be able to quickly or cheaply extract the information embedded in the IT artefact, as opposed to its surface design.

The model emphasised the interactive nature of innovation between individuals and firms and the importance of the public domain. The model also assumed the existence of bounded rationality and the potential for network externalities.

This model was then used to illustrate the role of intellectual property laws within the contexts of:

• confidential information;
• patent; and
• copyright.

The application to each intellectual property regime each suggested that the key to successful innovation policy was the promotion of information
flows so as to enhance the creation and distribution of tacit knowledge. For confidential information, the mechanism was competitive pressures encouraging reverse engineering. For patent, the grant of property rights encouraged disclosure of codified information which in turn generated learning. For copyright, the idea-expression dichotomy promotes access to ideas as the means for creation of new works.

The remainder of this thesis will examine one particular area of application of intellectual property law—copyright over computer software—and assess the economic justifications and legal determinations against this model of innovation. Chapter Five argues the proposition that software—computer programs—are an “IT artefact” within the meaning of the earlier discussion. The implications of this for innovation policy within the industry is discussed in Chapter Six, followed by an assessment of the current state of play within the international legal environment in Chapter Seven.
Chapter Five
Software as an IT Artefact

5.1 Introduction

The earlier chapters in this thesis developed a model of innovation based on information economics and the concept of an IT artefact. This chapter argues that computer programs distributed in machine code form only are IT artefacts as defined in Chapter Four. This requires the overturning of a widely held assumption in the copyright literature – that the technology underlying a computer program is readily appropriable. Support of the contrary proposition requires closer examination of the nature of computer processing and in particular the nature of programming languages. It also requires an understanding of the software development process. These matters are set out in sections 5.2, 5.3 and 5.4, with additional detail in the Appendices.

The key issue is one of access to the underlying technology by possession of the IT artefact. One way to put this question in the context of computer programs is to consider to what degree the possessor of executable or machine code (the IT artefact) has access to its source code (the technology). This access is widely assumed within the literature, but challenged in this thesis. The principal reason is that, for third and higher generation languages (defined below) source code is not isomorphic with machine code, nor does decompilation provide a
technological solution.

Having established this key proposition, the model of innovation presented in Chapter Four is then applied to software as an IT artefact, and some implications of the model discussed. In particular the model suggests a minimal role of copyright and a significant role in the encouragement of imitative competition for technological development. These implications provide the basis for the discussion of innovation policy which follows in Chapter Six.

5.2 Computers and programs

The term “computer” is capable of a variety of usages. The primary meaning is a short-form for an “electronic, digital, stored-program computer.” The term “computer” is often used ambiguously and has three main meanings:

• the complete apparatus;

• the electronic components only (including keyboard, screen and power supply); and

• in a narrow sense, the “central processing unit, (CPU) which is at the heart of the electronic componentry.

This thesis uses “computer” to mean a device which is capable of performing a defined set of operations on a series of binary digits (“bits”), hence, the device is presumed to be digital. Each operation results in a change to some part of the machine’s internal state. An operation is
performed when an instruction to do so is loaded into a part of the computer called the "instruction register".

The complete set of instructions that a computer is capable of executing is called the computer's "machine-language". The set of instructions that makes up a machine-language is designed to be sufficiently rich to enable the computer to perform a particular function. A "machine language program" is a program that is expressed in a particular computer model's machine-language. Because of the nature of the technology used in a computer, a machine-language program is a series of bits, conventionally represented in human readable form as either "1" or "0". A computer is capable of executing any program that is expressed in its machine-language. A computer is said to "execute" a machine-language program by loading successive machine-language instructions into its instruction register. The instructions that make up a machine-language program are accordingly often referred to as "executable code".

The word "hardware" is used as a generic term for computers and associated physical equipment. This is complemented by "software", the generic of program. Although the terms "program" and "software" can be used in a restrictive manner, to refer only to machine-language programs comprising executable code, they are more commonly encompass code that is expressed in languages other than machine language, but which is convertible into executable code.

Programming in machine-language is tedious and inefficient. One
important reason for this is that it is, definitionally, oriented towards the
machine’s internal structures, whereas, the purposes for which
organisations and individuals want to apply computers have their own,
different natural structures. Hence, for practical and economic reasons,
very little programming is performed in machine-language.

Instead, programs are usually expressed in some other language that is
capable of being converted into machine-language. Many languages have
been designed to fulfil two key requirements, to be:

• convenient and efficient for expressing whatever organisations and
  individuals want to achieve; and,

• capable of being translated into machine-language.

Unlike machine-language formats, which are expressed in binary code,
programs expressed in other languages use some form of “character-
representation code”. The most commonly-used is “7-bit ASCII
(American Standard Code for Information Interchange). This uses a
series of 7 bits to represent a set of 128 characters, including all of the
upper and lower-case letters of the Roman alphabet, the numerals 1-9 and
zero, and punctuation marks. An example of this is that the binary string
1000001 represents the letter “A”.

Other character-representation codes exist, including an 8-bit
international code, ISO 8859 which has the capacity for 256 characters and
can, therefore, support “diacritics” (qualifiers to the Roman alphabet such
as ü, ø, ç, and é) and a 16-bit code called Unicode, whose 65,000
combinations can also accommodate ideographic codes such as Kanji.

5.3 Programming languages

The term "low-level languages" is commonly applied to assembly languages and macro-assembly languages (second generation languages) and the term "high level languages" to third and higher generation languages.160

First generation languages ("machine language") were expressions entirely in terms of the computer's instruction set. Problem solutions were expressed in machine terms, not human terms. Over time these instructions were condensed from binary to octal and hexadecimal notation for convenience and efficiency.

Use of second generation languages ("assembly" and "macro-assembly") was a superior programming technique since these languages involved the use of mnemonic commands rather than arbitrary numeric commands. Problem solutions were still expressed in machine terms since the assembly languages were isomorphic with the underlying machine languages.

The significant advance in computer programming was the emergence of third generation languages ("algorithmic" or "procedural") which enabled problem solutions to be expressed in a form more logically transparent

---

160 This overview of software generations is principally drawn from Clarke, R. (1991) "A Contingency Approach to Application Software Generations" Database, Summer.
than machine or assembly language. The difficulty in development of third generation languages was the breaking of the isomorphic connection between the language and final machine code executable which had existed in the second generation. Third generation languages emerged in the early 1950's with major theoretical advances association with Fortran (“formula translation” language), Algol (“algorithmic language”) and COBOL (“COmmon Business Oriented Language”). In more recent times, languages such as Pascal and C have dominated the creation of programs for microcomputers.

The fourth generation has been conceived as languages whose defining characteristics include productivity, end user development, relational data modelling, and non-procedural programming. This final characteristic has been the most widespread perception of fourth generation languages. Manifestations of fourth generational movements have been interest in reusability, object-oriented programming languages and application generators.

The emergent fifth generation represents an interest in logical programming, expert systems, and artificial intelligence. Simplification of problem solving can be linked with the engineering of human expert knowledge within the computer architecture. This involves the creation and maintenance of a pre-defined knowledge base which includes the use of rules supported by reference data. The intention is to express a domain-specialist’s knowledge about a particular problem domain. The
theoretical sixth generation would be a further level of abstraction from
the machine limitations. Table 5-1 summarises the abstractions
interpretation of application software generations.

Table 5-1 – Application Software Generations

<table>
<thead>
<tr>
<th>Language</th>
<th>Human Activity</th>
<th>Machine Activity</th>
<th>Man-Machine Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Facilitative</td>
<td>Collection and provision of empirical evidence</td>
<td>Development of implicitly pre-defined domain model</td>
<td>Delegation by an operative to a “black box”</td>
</tr>
<tr>
<td>5 Descriptive</td>
<td>Definition of a domain model</td>
<td>Application of implicitly pre-defined problem solution</td>
<td>Delegations by a teacher to a “black box”</td>
</tr>
<tr>
<td>4 Declarative</td>
<td>Definition of the problem to solve</td>
<td>Application of an explicitly pre-defined problem solution</td>
<td>Delegation by a teacher to an educated clerk</td>
</tr>
<tr>
<td>3 Algorithmic or procedural</td>
<td>Expression of the problem solution in a natural form</td>
<td>Non-isomorphic translation (compilation and interpretation)</td>
<td>Instruction by a teacher to a dumb clerk</td>
</tr>
<tr>
<td>2 Assembly or macro-assembly</td>
<td>Expression of the problem solution in mnemonic shorthand</td>
<td>Isomorphic translation (assembly)</td>
<td>Use by the specialist of an extended tool</td>
</tr>
<tr>
<td>1 Machine language</td>
<td>Expression of the problem solution in the machine’s terms</td>
<td>Computation</td>
<td>Use by a specialist of a tool</td>
</tr>
</tbody>
</table>


The term “program” might now be defined. It is conventionally used only to refer to sets of instructions that are expressed in a language of generations 1-3 inclusive, i.e. a machine-language, an assembly-language
A program is therefore an expression of an algorithm that is designed to cause a computer to perform a particular function. The term "instruction" is used to refer to each statement; whereas, later-generation languages comprise statements that are not suitably referred to in that manner.

Because the approaches are still relatively new, there is no conventional term for blocks of statements that express problem-descriptions (fourth generation) or problem-domain-descriptions (fifth generation). What is, and what is not, a "program" is technology dependent and context driven.

The term "source code" is used to refer to the statements that make up a program, irrespective of the language that is used – source code may be in machine-language, a low-level language or a high-level language.

All programming languages are formally defined, with a set of rules defining the specific "instruction set" and "language syntax" that are to be used. The sets of instructions available in most languages are rich. The result is that, for any given function that needs to be performed, there are generally a number of ways in which the program can be expressed. For complex functions, there may be a myriad of different expressions that are functionally equivalent.

---

To simply the process of program development, coding, and testing ("debugging"), computer programs have been written in preliminary forms which are far more convenient for human scrutiny. "Source code" does not constitute a set of instructions: in that source does not direct the computer to perform a function. Source code is an outline of the steps to be taken in the solution to the problem forming part of the program. Source exists in varying levels of abstraction and sophistication and framed around "languages" which are, in a broad sense, symbolic statements in human-convenient form: for our purposes, a form representing English.

When a program is expressed in a language other than machine-language, it needs to be processed by a "translator". The output from a translator is referred to as "object code" – where "object" is user in the sense that this was the "object" or "objective" of the translation activity.

The object code that is produced by a translator is one of the following:

- executable code expressed in machine-language;
- machine-language that is not yet executable, because it needs additional elements to be associated with it, which are drawn from a library of common components. The process involved in associating those additional elements is known as "linking; or,
- an intermediate form that requires further translation before it is capable of being executed.
Translators are of various kinds, depending on the kind of language that they translate, and when they are designed to be used. The classes of translators are:

• for a translator that takes as its source code input a complete program:
  * if that program is expressed in a 2nd generation language – an "assembler";
  * if that program is expressed in a 3rd or later generation language – a "compiler";

• for a translator that reads as input not a complete source code program, but merely the next statement in a source code program and provides the output for immediate execution – an "interpreter".

Note that this approach can be used for any level of language.

Combinations of these techniques are also possible. In particular, a high-level language may also be compiled into intermediate code, that is subsequently handled, instruction-by-instruction, by a "run time interpreter". This enables programs written in a high level language to be run on different models of computer, provided that a run-time interpreter is available for each of the different machine-languages.

The complexity of the translation process undertaken by compilers is generally much greater than that for assemblers, because the structures of the source code and the object code are so radically different from one another. The richness of the languages and the options that are available
to the people who construct compilers, are such that different compilers, designed to translate the same source code into object code for the same computer, may produce significantly different object code.

A translator can only produce effective object code if the source code:

- complies with the formal specification of the source code language;
  and;
- expresses the function that the designer intended.

Translators generally test for compliance with the language syntax and report errors back to the programmer. They cannot, however, test for compliance with the designer's intent, because there is no external reference point with which to compare the source code or object code. This is an expression of the linguistic distinction between "connotation" (meanings and emotional states evoked by the words) and "denotation" (literal meaning). Programs are only capable of understanding denotation.

The majority of software is written in languages of the first three generations, hence, close attention needs to be paid to the manner in which programming is performed in these languages. Languages of the first three generations are referred to as "procedural" or "algorithmic" languages. By this it is meant that a program comprises a precise formulation of a plan (or procedure or algorithm) for scheduling actions to be performed by the computer. Examples of algorithms in everyday life would include: directions for reaching a particular destination by road; a
cooking recipe; and the procedure for calculating the gross wages for a particular employee. A verbal representation of an algorithm is presented in Figure 5-1.

**Figure 5-1 – Payroll Algorithm**

| If the hours worked are less than or equal to 40, the pay is the product of the number of hours worked and the rate of $10.00 per hour. Also, if more than 40 hours are worked, the pay is $15.00 per hour, for each of the hours over 40. If more than 60 hours are worked, the pay is $20.00 per hour for each of the hours over 60. |

A representation of this algorithm in the programming language C is given in Figure 5-2.

**Figure 5-2 – Payroll Algorithm in C**

```c
/* Example payroll algorithm */
#include <stdio.h>
main()
{
    int hours, pay;
    scanf("%d", &hours);
    if (hours <= 40)
        pay = 10 * hours;
    else if (hours <= 60)
        pay = 10 * 40 + 15 * (hours - 40);
    else
        pay = 10 * 40 + 15 * 20 + 20 * (hours - 60);
    printf("Gross pay is %d", pay);
    return 0;
}
```

Algorithms may be “foolproofed”, or made “robust” by adding additional procedures to deal with anticipated errors in the input process. For example, if the program in Figure 5-2 receives alphabetical input rather than numeric, or if the total hours worked are negative or greater than

---
162 This example based on Metrowerks Inc., (1995) *Codewarrior: Principles of* page 95
some maximum level. The program could respond by the display of an appropriate error message or "alert".

An important feature of algorithms is that two algorithms which appear to have substantially different structures may exhibit logical equivalence. For the same inputs, each algorithm will produce identical output. To demonstrate, take the algorithm in Figure 5-2 as one potential implementation of the verbal algorithm in Figure 5-1. The distinctive process involved is that the gross pay is calculated by way of a single computation using a different formula for each category. An alternative implementation is presented in Figure 5-3.

Figure 5-3 – Alternative Payroll Algorithm

```c
/* Alternative payroll algorithm */
#include <stdio.h>
main()
{
    int hours, pay;
    scanf("%d", \&hours);
    pay = 10 * hours;
    if (hours > 40) {
        pay = pay + (5 * (hours - 40));
    }
    if (hours > 60) {
        pay = pay + (10 * (hours - 60));
    }
    printf("Gross pay is %d", pay);
    return 0;
}
```

This second algorithm instead makes increments to the basic wage. If the two pieces of source code are compared, they are distinct in expression. They are also distinct in mathematical process. When applied to any

*Programming* (Metrowerks) at pp. 45 - 63.
given set of inputs, however, the output will be identical. This is a trivial
demonstration of the general proposition that, for any set of executable
program behaviours, there are a range of potential source code strategies
which can generate the same behaviour.

5.4 Software development

A software development project will typically include a number of input
files which are collectively described as the project's "source code", even
though only a small part of the project will be newly written source code
in a higher level language. Central to this complex software development
environment is a project file. Some application development
frameworks use a project file to keep track of all the relevant source
materials and object code, as well as specifications of the procedures
through which the project is to be conducted, such as specifying the
targeted operating system or setting optimisation parameters. Some of
the principal components of a project will be source code files which are
text format files containing parts of the program algorithm represented in
a higher level language form such as C, C++, Pascal, or BASIC or a lower
level form of Assembler language. Depending on the particular language
used, these source code files may also include interface files or header files
which are other text format files containing definitions of the data and
functions used in the source code. The compilers for some languages
such as Pascal and C require formal specifications of functions and
parameters used in the source code.
Another element of project source materials are resource files which are descriptions of items used within the source code, particularly in a graphical user interface environment. A typical resource file will contain descriptions of windows, dialog boxes, menus and other graphical items which may have been constructed using a special application. Resource files may be in binary (object) format or in a text format requiring a special resource compiler to place it in binary format before being linked.

All text format files, whether source code or resources, must be transformed into a binary form. The resultant code is described as object code. To obtain object code, the source code is “run” through a compiler which takes the higher level language expression and compares it with a predefined vocabulary, which is a cross-referencing of the higher level language expressions with their binary equivalents. Source materials will also include library files which are precompiled elements of source code which are common and reusable between different software projects. This avoids the need to rewrite source code such as that representing elements of the user interface (such as MacOS or Windows libraries). If part of the source materials are in Assembler language, then the object code will be produced using an assembler rather than a compiler.

Once the project has been compiled or assembled into object code, the final stage is for a linker to take the object code and link the pieces together to form the desired software product. The most obvious product is an application or executable which is a file directly executable by the computer. The linker will ensure that the object code is compatible with
the central processing unit (CPU), such as those based on the Intel 80486 chip used on IBM PC's, or the Motorola 68040 used on the Apple Macintosh. This process is illustrated in Figure 5-4.\textsuperscript{163}

\textbf{Figure 5-4 – Compilation Process}


This process is presented in more detail in Appendix One, which also provides the basis for examples in further appendices.

Once the algorithms have been coded, the program should be tested using an appropriate strategy based on comparison of a range of inputs (including deliberate errors to ensure that the program is robust) to expected outputs. If there are differences, the sources of the errors are investigated. An error may lie in the coding process, the algorithm itself, the design and structure of the program, or the computation of the expected results. Serious errors in the program are usually obvious, such as the program “freezing” or “hanging” (execution stops or is continuously looped) or “crashing” (the program quits and/or damages the systems software).

If an error lies within the coding process, the strategy for identifying and removing these errors from the source code is described as debugging.
This process is assisted by a range of specialised applications. A debugger is a particular application which simulates and controls the execution of a program so that the programmer can observe the program running and the changes which are occurring internally. The appropriate debugger can execute the program statement by statement, suspend execution when it reaches a certain point, or interrupt the program if some particular value is reached in a memory location. The debugger allows the programmer to view the functions, examine and change the values of variables, and inspect the contents of the processor registers. Many debuggers allow the programmer to view the source code in its assembler form as well as in the source code equivalent. Some also facilitate editing of the source files during this process.

To assist in the error detection, debugging and future maintenance and enhancement of programs, it is good practice to annotate the source code extensively. These annotations, which are themselves lines of code, are described as “comments”, and are ignored in the compilation process. The comments should record a detailed explanation of the source code and its intended effect. This is particularly useful where different programmers may be asked to update or rewrite the code, such as the incorporation of new features in an existing application. Source code without comments is difficult to interpret, particularly within lengthy source code for a complex application. The example in Appendix Two demonstrates the usefulness of commented source code in providing a basis for understanding a program.
5.5 The Issue of Access

Prefaced by the above discussion, this section argues that software is an "IT artefact" in the terms developed in the previous chapter. A key issue is the degree of access or "appropriability" exhibited by software. In terms of the model proposed by this thesis: 164

The significant characteristic for the purposes of the model in this thesis is that that the knowledge underlying the creation of [software] is not readily discernible to the person using the [software] . . . . The information (or technology) underlying its creation will typically be difficult or impossible to access merely from its use.

Such a proposition is at odds with a significant proportion of software copyright literature, which assumes that distribution of the program entails distribution of the ideas underlying its creation. 165 This widely accepted - yet flawed - assumption in policy articles about computer program copyright is the argument that computer programs reveal a substantial amount of knowledge about the means of their creation on the "face" of the product. This section argues that such a degree of access is, however, a myth. Distribution of an application does not also mean disclosure of the means for creation of that application.

One can use an application, or interact with an interface without understanding how it works. Users, for example, of Microsoft Excel may become proficient in spreadsheet design, and in the use of formulae and

---

164 See Chapter Four, above.
macros. This does not mean that the same user is capable of constructing a spreadsheet application. It may provide some clues as to potential ways of implementing such an application, but learning from clues is the essence of access to ideas which copyright intends to promote.

One fallacy lies in confusing copyright doctrine with the evidentiary issues involved in proving infringement. A common theme is that new technologies will permit copying in such a way as to avoid detection or the appearance of similarity between works.\textsuperscript{166} A plaintiff must prove access and substantial similarity. A major legal consideration should be to what form of the program the alleged infringer had access. The common assumption by commentators that "access is generally not an issue"\textsuperscript{167} assumes one or more of the following:

- that the infringer had access to the source code;
- that source and machine code are interchangeable; or
- that technology exists to translate machine code into source code.

The following sections examine each of these cases in turn.


5.5.1 Access to Source Code

One source of the misplaced assumptions regarding access arises from the fact that historically some programs were distributed in source code form. For example, Harris\textsuperscript{168} noted that the most likely scenario for reverse engineering of a computer program is by distribution of the program in source code form, or in assembler form with heavy comments such that the assembler language is readily accessible. Rilee\textsuperscript{169} also assumed that both source and object code were commonly licensed and thereby misappropriated.

Both these commentators were writing in the early 1980's and their experiences based on practices in the 1970's principally dealing with mainframe and minicomputers. Such a situation no longer exists. For the microcomputer programs of today, it is extremely rare for software to be distributed in source code. As a practical matter, programs are distributed in a form which is understandable by computers - in the sense it is executable by the hardware and any system software - but not by human beings. This is particularly true of mass marketed software for microcomputers. For mass marketed or "shrinkwrapped" software, the source code is not revealed, and the distribution takes place only in

\textsuperscript{168} Harris, J.R. (1985) "A Market-Oriented Approach to the Use of Trade Secret or Copyright Protection (or Both?) for Software" Jurimetrics Journal, Vol. 25, p. 147 at p. 160.

5.5.2 Access to Machine Code

Another source for the assumption of access is that machine code and source code are interchangeable. McGahn\textsuperscript{170}, for example, incorrectly described a compiler as "nothing more than" a program which "converts" source code into machine code. Portraying executable code to be a translation of source code in the same manner as English translates into French, would be misunderstanding the nature of programs. Viewing machine code as a "mere translation" of source code promotes the misconception that compilation is isomorphic, and confuses the artefact with the knowledge underlying the artefact. Such misconceptions are evident in assertions by commentators such as Davidson that "reverse engineering" of a machine is required to understand the means of its creation but an understanding of a computer program is obtained simply by "reading" the computer language.\textsuperscript{171}

In part this incorrect assumption may also be grounded in legislative and judicial history. The first case in Australia,\textsuperscript{172} for example, concerned the


protection of a ROM chip which only involved assembly code and executable code. Similar cases were considered in the United States.\textsuperscript{173} As the expert evidence led was limited to the case at hand no consideration of any more complicated processes was necessary. This set the ground (in Australia) for a conception of computer programs as merely being constituted of "source" (assembler) and "object" (executable) code – one being an identical translation of the other. This relationship is illustrated in Figure 5-5.

Translation of source materials into machine code is "mechanical" only in the sense that it is carried out by a machine. When dealing with third and higher generation languages, it would be incorrect to compilation this as an "isomorphic" translation in the sense of an identical mapping of one to the other. In discussion the compilation process, Davidson incorrectly asserted:\textsuperscript{174}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure55.png}
\caption{ROM Cases}
\end{figure}

\begin{itemize}
  \item Source Code (Assembler)
  \item Machine code (Executable)
  \item Rom etcher (Physical translator)
  \item Isomorphic translations
\end{itemize}

\begin{itemize}
  \item Comparison of the American and Australian Decisions” \textit{U.N.S.W. Law Journal}, p. 143.
  \item \textsuperscript{174} Davidson, D.M. (1983) “Protecting Computer Software: A Comprehensive Analysis”
\end{itemize}
The code has changed; the ones and zeros of object code are very different than the letters, numbers and symbols of source code. But the expression remains the same; only the size of the alphabet has been reduced.

Davidson's statement suggesting inherent isomorphism between source and object code would only be valid in respect of the relationship between assembler and machine code, and would be invalid in respect of a third generation language (such as Pascal).

As discussed above a computer program is merely a mechanised manner of solving a problem expressed in the form of an algorithm. Logically, as there may be more than one means of solving a problem, more than one algorithm may serve the same purpose. Samuelson et al.\textsuperscript{175} noted that two programs with different texts\textsuperscript{176} can have equivalent behaviour. To a large degree, in programming, the algorithm selected will depend on the:

- tacit knowledge and inherent problem-solving ability of the originating programmer or programmers;
- computer language selected for the original and the recompiled program;
- the compiler program and the program used to decompile;
- amount of time available to complete and test both the original


\textsuperscript{176} Samuelson \textit{et al.} use the examples of Lotus 1-2-3, VP-Planner and The Twin.
program and the program to be recompiled; and,

• processing capacity and speed of the processor on which both the original and the recompiled program is to run.

Given the functional equivalence of algorithms, it follows that the same function can be expressed by a range of distinct algorithms. Identifying the precise algorithm, expressing the internal logic of the source code, is for any sizeable or complex program, an impossible task if only the machine code is available. The proposition that there is equivalence between source and object code is incorrect. Mere access to machine code does not imply access to the source code.

5.5.3 Access by Decompilation

Some of the copyright literature assumes that technological means exist to facilitate access to the underlying source code. The recovery of lower level source code is referred to as "disassembly". The process of conversion into higher level language source code is referred to as "decompilation". The concept of decompilation raises the question as to whether the particular source code that gave rise to a particular program in machine code can be recovered from the executable code with the aid of a decompilation program. The literature is replete with statements about the ease of access to source code by means of decompilation. Davidson177

asserted, for example, with no evidence that:

It is possible with many computer languages to reverse object code back into a form of source code. This quasi-source code would not necessarily look the same as the original source code, but would contain *virtually* the identical specific logic and design which is the expression of the program.

The use by Davidson of the words "virtual" and "identical" significantly distorts the true situation. Similarly, Walter\textsuperscript{178} proposed that, given training in the relevant syntax, humans could understand "the information expressed in any computer language". More recently, Samuelson *et al.*\textsuperscript{179} stated that:

The object code in software packages can, through a decompilation process, be translated into a form that approximates to some degree the source code that the program’s developer maintains as a trade secret.

These assertions are misplaced. Whereas executable code is particularly prone to direct replication, it is extremely unlikely that the results of disassembly or decompilation will resemble the original source code, even in terms of its general "logic and design". Trotter Hardy\textsuperscript{180} noted that one problem with the analogy of "translation" applied to programs is that translation is usually reversible. Computer program translation


from source to object code is not reversible, despite the widespread assumption that decompilation provides a technological vehicle for reconstruction of the original source code.

This is due to several factors such as the large number of functionally equivalent expressions that could possibly have produced a given executable code program. This lack of isomorphism between source and object code has been discussed above. Further, to approximate functionally equivalent code, a programmer would need to know not only the precise version of the particular translator that was used to produce the object code, but also the full internal details of the design features of that translator as well the precise versions of any other components of object-code that were linked together to make up the executable code (see sections 5.3, 5.4 and Appendix One).

Another difficulty is encountered in attempting to reconstruct the original source code. In many modern programs, comments lines entered into the original source code for the purpose of legibility to a human reader are ignored and redundant code is discarded as part of the compilation process – so do not form part of the final software product. The importance of comment lines in understanding the structure and rationale and the source code is discussed above and illustrated in Appendix Two. Any commercial software project would require extensive annotations in the form of comments, as well as documentation of the design and implementation process for successful
maintenance, debugging, and upgrading.

At a practical level, decompiled code is typically at a low level and decompilation software can at best translate the set of instructions contained in the final binary into an assembler form. The word "decompile" is typically used incorrectly to suggest a reversal of the compilation process. In many instances the word "disassemble" is more appropriate. When a program is "decompiled", the result is typically code resembling an assembly language. The algorithms within a decompiled program can only be represented at a very low level of abstraction – the set of instructions to the CPU to make specific calculations and to deal with variables identified by register. Assembler is based on a set of basic calculation directives to the CPU based on values contained in registers, as well as comparisons and copying of values between CPU registers and other locations in memory. In such a form, the decompiled code will at best be cryptic and at worst be unintelligible. Often, the content and interrelations of these registers remain uncertain. For programs of any size or complexity decompilation will offer few clues as to the nature of the original source materials.


Samuelson et al. argued that, given access to decompilation software, not only surface design but also internal design lies “near the surface” of a software product. This proposition is incorrect for third generation languages and particularly unjustified in respect of higher level languages or where sophisticated compilers and optimisers have been used to produce the executable code. Increasingly sophisticated optimisation procedures and intelligent compilers and linkers also can result in the final binary code bearing no resemblance to the original text based work of the programmer. It is difficult if not impossible to determine whether a program has been optimised either generally, or specifically for a particular CPU architecture. Optimisation may even rearrange sequences of instructions sent to the CPU to draw upon that particular CPU’s strengths so that processing time can be minimised. Such optimisation is typical in reduced instruction set computing (RISC) environments, such as the PowerPC architecture. Daughtrey’s description of the usefulness of decompiled code is the most accurate and appropriate of the commentators in the area:


Reading decompiled code is comparable to reading a novel that has been stripped of all adjectives, adverbs, articles, and other explanatory words; reorganised to be completely chronological with no chapters or paragraphs; and changed so that the characters, places, and other nouns are represented by a single letter followed by a single digit.

This description is supported by Appendix Three which provides an example of a small section of decompiled machine code, extracted from the binaries in the SillyBalls project illustrated in Appendix One.

Decompilation has been assumed to be able to create a competing integrated product. But in programs of any significant size or complexity, such a task would be impossible or at the very least prohibitively costly. Reverse engineering of a computer program using decompilation will usually be more difficult than writing a program from scratch. The usefulness of decompilation more typically lies in close analysis of some small element of the executable program with a view to error correction or, more commonly, interoperability with other software and interfacing with communications hardware. A person seeking to write a competing program would benefit more from possession of the manual rather than a decompiled program. The assumption that decompilation provides access to the source code of a program is incorrect.

5.6 A software innovation model

Section 5.5 demonstrated that software met the key element in the definition of an "IT artefact" from Chapter Four – that the artefact does not of itself disclose the means of its creation. It should be noted that this will only be true in respect of programs which are distributed in machine (executable) code alone. This reflects, however, the nature of software commercially distributed in the consumer and desktop business markets.

Given that such software agrees with the concept of "IT artefact", the innovation model presented in the Chapter Four can be adapted to reflect the programming process set out in this Chapter. This programming innovation model is presented in Figure 5-6.

5.6.1 Explanation of the model

Tacit knowledge in the model resembles an abstract solution algorithm applied to solve a particular programming problem; it would also include the know how arising from programming expertise and technique appropriate to a particular hardware architecture and systems software. The codification process, or "articulation" of this solution is the creation of software project source materials. While part of these materials would be "source code", in the sense of new third or later generation language code written for the project, they would also include libraries, reusable code from other projects, and the collection and integration of these materials within a project file or application framework. The software development process described above in section 5.4 and illustrated in
Figure 5-6 - Software and Innovation
detailed in Appendix One would be applied to these source materials to produce the machine or executable code application. This accords with the concept of codified knowledge as a means of production. The application in its executable form is the IT artefact, the form in which the software is distributed to the marketplace. Most microcomputer applications are licensed and distributed only in their machine code forms. Since the compilation process is non-isomorphic, the application does not convey to users an understanding of the source code. At best the machine code can be decompiled into an assembly language. Reverse engineering and cumulative innovation will occur through use of the application and observation of its execution. The programming ideas at a high level of abstraction can be used by other developers in the construction of their own applications. Computer programmers create improved programs by studying the work of earlier programmers.186

5.6.2 The role of copyright

The model of copyright discussed in section 4.4.3 is not the appropriate model of innovation given the above discussion of access. The copyright model assumed a joint product (represented in blue in Figure 4-5) which combined the characteristics of codified knowledge with the artefact itself. In this model, possession of a protected work permitted appropriation by users who did not contribute to the innovation process. In contrast, the

principal proposition in this chapter is that program users do not have access to the codified knowledge (source code).

Given this lack of access, the policy objectives of copyright would appear to be in conflict with the nature of the technology. This proposition is supported by writers such as Kreiss who argued that, without copies that can be perceived and studied by individuals, society was not getting the kind of access that is at the heart of the copyright system:

For competing programmers, the barrier to access to computer programs imposed by distribution in object code serves to thwart copyright’s traditional objectives. In more traditional contexts, a competing author who wishes to learn the writing techniques in a published novel or a poem can purchase the work and study those techniques.

Academics who filed an amicus curiae brief in the case of Sega v Accolade rejected the assumption that public use of the work equated with the objectives of the laws:

This argument incorrectly assumes that the only public purpose served by access is consumer use of the work in question. In fact, distribution to the public for consumption has nothing whatever to do with the idea/expression distinction or the goals of section 102(b) [which] are aimed at defining what later authors can freely use in creating new works.

The legal effect of a broad scope of program protection would be that copyright holders would be able not only to maintain the source code as

---


confidential information, but also to exploit that competitive advantage for a period significantly longer than the natural lead time would permit. Samuelson\textsuperscript{189} noted that CONTU\textsuperscript{190} believed that providing copyright protection over programs would reduce reliance on trade secret laws and provide a greater flow of information through the programming community. In fact the evidence suggested that the industry had no intention of abandoning secrecy as a means of competitive advantage.

But for one characteristic of computer programs, the above discussion suggests that a copyright regime is unnecessary to promote innovation. Since, however, the machine code form of a computer program is easily and cheaply replicable, there will be market failure which should be corrected by a legal rule against such replication. The basic argument from traditional analysis is that “software is expensive to create but easy to copy”.\textsuperscript{191} The economic impact is a lack of sufficient lead time to recoup research and development expenses. This argument explains and justifies, to a degree, the application of copyright law to literal or near-literal copying of computer programs. Machine code is easily replicated. Independent creation of identical machine code is extremely unlikely.

\begin{itemize}
\item[190] The United States’ National Commission on New Technological Uses of Copyrighted Works.
\end{itemize}
Prohibiting the replication of machine code without permission of the its creator corrects a potential market failure, and provides an incentive to invest in program innovation. It is the ease of replication of digital media\textsuperscript{192} which posed significant challenges to intellectual property laws, and generated initial policy moves towards copyright protection of software.\textsuperscript{193} Copying of machine code represents appropriation of the program’s behaviour at no cost and with no independent development effort or learning process. This represents the most serious danger of market failure in the absence of regulation.\textsuperscript{194}

5.6.3 Imitative competition and lead time

The implication of the innovation model to programming suggests that it is highly analogous to the confidential information scenario discussed in section 4.4.1. The tacit and codified knowledge, including project source materials, are represented in red on Figure 5-6 as being protected by secrecy and confidentiality within the bounds of the firm. Only the application in machine code form is exposed to the market and from its

\begin{flushleft}
\end{flushleft}

\begin{flushleft}
\end{flushleft}

\begin{flushleft}
\end{flushleft}
use and observation of its functions other firms will initiate projects of their own which are intended to compete with each other. This involves a process of learning by imitation in an environment where an innovator will be protected by natural lead time. Competing firms will create their own project source materials and in doing so, expand tacit knowledge.

Some commentators\textsuperscript{195} rally in favour of broader protection of non-literal elements on the assertion that the software industry is "unique" is having a short time between release of one product and the emergence of imitators. These same commentators often also assume a substantial research and development phase underlying one product and negligible time and effort for imitation. Samuelson \textit{et al.} argued that information products such as computer software:\textsuperscript{196}

\ldots bears so much of the technical know-how required to make them on or near the surface of the product that natural lead time for this kind of industrial product may not suffice.

The previous section has rejected this proposition where a program is distributed in machine code form only. As Daughtrey\textsuperscript{197} stated:

\ldots this form of a computer program, unlike most literary works, does not disclose on its face the ideas underlying the work.


Lead time is not eroded in a market destructive way. The evidence is widely accepted that, for computer hardware, primary vendors have had a lead time advantage since competitors needed to study the product after release to the marketplace and engage in the non-trivial task of development of compatible and competitive products. This lead time brings with it more enduring competitive advantages such as reputation, product quality and price. Brand recognition for earlier innovators has been found to be particularly significant. In an industry where rapid obsolescence is increasingly the case, by the time a second developer’s product is released to the market, the first developer may have already completed an upgraded version. Combined with brand loyalty, such a situation provides a natural reward to the first innovator.

A similar situation is present within the software industry. Evidence supporting this supposition is that in the United States, the program industry grew without any direct intellectual property protection from an output of $450 million in 1969 to $3,200 million by 1974. The fact that a large amount of growth can occur without exclusive proprietary rights may indicate that innovation is also rewarded through other factors such

---


as the first mover advantage,\textsuperscript{201} or may be motivated by a fear of losing market share to competitors who innovate.\textsuperscript{202} Market lead time may be significant in light of the rapid obsolescence of programs.\textsuperscript{203} Breyer has suggested that lead time alone may be sufficient for program manufacturers to recoup their investment.\textsuperscript{204} Such arguments have typically been rejected as naive.\textsuperscript{205}

Menell\textsuperscript{206} outlined other features of the software industry by which software developers could internalise the public goods problem. One was by the use of licensing agreements with customers prohibiting reproduction and dissemination. Due to policing problems,\textsuperscript{207} such strategies are more effective in software written for mainframes and

\begin{thebibliography}{99}
\bibitem{203} Multilateral Trade Division, Department of Foreign Affairs and Trade, \textit{Intellectual Property Rights Problems in Overseas Markets: Summary Report} (July 1988) p. 11.
\end{thebibliography}
minicomputers rather than microcomputers.\textsuperscript{208} Another technique was
the prevention of replication through technological measures including
"software locks".\textsuperscript{209} Other solutions included use of research consortia to
spread development costs and share the benefits of new technologies.
Government research expenditure, particularly on space and defence
technologies in the United States, and indirectly through funding
universities has also been significant in promoting invention. Firms can
also reap benefits by providing associated services to a software product,
such as installation, training, maintenance and strategic management
consulting.

The phenomenon of computer programs applied to the innovation
model suggests that copyright – if it is to govern the process of innovation
in the software industry – should sanction of the process of learning by
imitation, where competing firms obtain ideas from the use of other
applications. These issues of policy are discussed in the following chapter.
As discussed in section 4.4.1, this task of reverse engineering unpatented,
undisclosed innovation contributes to the innovative community's
overall costs of research and development. Reverse engineering provides

\textsuperscript{208} Harris, J.R. (1985) "A Market-Oriented Approach to the Use of Trade Secret or
Copyright Protection (or Both?) for Software" Jurimetrics Journal Vol. 25 p. 147; and

and Sacks, J. (1985) "To Copy Protect or Not to Copy-Protect?" Popular Computing
(October) p. 73.
originators with natural lead time. Only followers who are prepared to defray the costs of study, analysis, and relearning will be able to exploit the cost reductions, technical improvements and potential new applications arising from the reverse engineering process – Reichman’s concept of “self-help diffusion”\textsuperscript{210}.

5.7 Summary

This Chapter has analysed the nature of a computer program. In particular it has outlined the nature of the programming process, including languages and algorithms, development of source materials, compilation and debugging.

The key issue for this chapter was the proposition that software is an IT artefact within the definition developed in Chapter Four. Under this definition, computer programs (distributed in machine code only) do not reveal their underlying technology (source code). Section 5.5 demonstrated that widely held assumptions about “access” were misplaced. One important aspect of this argument was a demonstration that codified information relating to programs is not revealed to users through use of the artefact and is markedly altered by the process of compilation and decompilation. Treating source code as a mere translation of executable code has led to the misconception that source is revealed through possession of the executable code. Decompilation is of

\textsuperscript{210} Ibidem p. 2522.

By perceiving decompilation to be a simple process, a program can be seen to suffer from appropriability problems, even when distributed only in machine code form. This perception aids the presumption that information contained in programs is easily misappropriable.

Having established that software has the characteristics of an IT artefact, innovation in the software industry was adapted from the model of technological innovation in Chapter Four. The close analogy to the scenario of confidential information suggested a role for intellectual property law in the encouragement of learning by imitation. The nature of programs in executable form did, however, imply a basic level of protection against replication of machine code.

Adopting this analysis, Chapter Six considers the justification of the extension of copyright laws beyond this basic level of protection. This is followed by an assessment of the current state of play within international copyright regimes.
Chapter Six
Implications for Copyright

6.1 Introduction

The previous chapter analysed software within the context of the model of innovation outlined in Chapter Four. This model was developed from a foundation of information economics. One message from this perspective is that the flow dimension of information is critical within an institutional framework of property rights governing innovation. Property rights which support, rather than restrict, information flows are essential in encouraging innovation. Chapter Five established that software distributed in machine code form required only a minimal level of protection against replication to encourage development. This was based on a model of imitative competition where natural lead time and other economic advantages support innovation rather than monopoly rights granted through intellectual property laws.

This chapter examines the issue of the appropriate scope of copyright protection beyond that of replication – “literal” or “near literal” copying – to find that little justification exists for extending protection beyond this basic level. Software is both a physical object and an abstraction. The physical object, although not humanly perceptible, does rest in electrical patterns within computer storage media. The abstraction is the algorithm and/or function on which the program is based. One difficulty for
economic analysis and the application of economic policies to legislation and cases is the psychological tendencies to extrapolate the abstract to the physical and the physical to the abstract. Davidson\(^{212}\) describes this as the "Cheshire cat grin" fallacy:

[S]oftware is not seen as the cat itself (a copy of the software) nor the unique pattern of stripes which identifies the cat (the specific design and logic of the program); software is viewed as in between, like the grin – not as tangible as the cat but not intangible either, since software can operate a computer.

One consequence of the "Cheshire cat grin" fallacy is the argument that a program with no objective similarity to another program may yet infringe copyright on the basis of reproduction of "non literal" elements. McGahn\(^{213}\) equates this term with the "intangible", "stylistic"\(^{214}\) and "creative" aspects of a program. Based on the model of innovation developed in the earlier chapters, and a critical analysis of the literature, this form of infringement carries serious dangers to the maintenance of information flows and reflects a treatment of information as a mere commodity.

The chapter begins with a critical reflection on the economic and creativity arguments favouring broader protection for software. The


\(^{214}\) Hobbs, P. "Methods of Determining Substantial Similarity in Copyright Cases Involving Computer Programs" University of Detroit Law Review, Vol. 67, p. 393 at
economic arguments are based on an extension of the commodity perspective outlined in Chapter Two. The creativity arguments misconceive the nature of the computer programming as akin to a literary or artistic endeavour. This leads to a consideration of the major policy issue in the literature and in the case law – protection of standards and user interfaces.

6.2 Economic arguments

As discussed in Chapter Two, the traditional economic model of informational artefacts is one which emphasises appropriability problems, and as a question of policy would seek to create “vendible commodities” out of intangibles. Chicago writers are particularly prone to this type of analysis. Dam, for example, argued that copyrights were:

... best thought of as property rights...

and that:

... economic concerns can be better assessed once we accept that the appropriability problem must be solved by some form of property

p. 396.


218 Ibidem p. 335.
As a result, limitations of the exclusivity of intellectual property rights—such as compulsory licensing and fair use—are seen as severely detrimental on the incentive to invest and the operation of the market mechanism in allocating resources. Conventional wisdom asserts that stronger intellectual property rights will inevitably increase the pace of invention, induced by the greater rewards available to the original inventor.

In discussing computer programs, Yanaga argued that social welfare is maximised where copyright awards the inventor a property right that is equal to the software's entire marginal value to society. Such sentiments are echoed by Palmer's assertion that by setting a longer term of copyright protection, there is a greater incentive to work on software with greater durability. Palmer's explicit assumption was that an expansion of intellectual property would always generate additional incentives to invest in additional software development. Similar sentiments were


evident when the decision of the US Congress to provide protection for works of architecture\textsuperscript{223} was justified in part on the basis that the monopoly incentive would “stimulate excellence in design”.\textsuperscript{224}

Economic policy in this arena is influenced by the argument linking a high degree of expenditure or commercial value to various parts of a program to justify protection. Commentaries on the United States case of \textit{Whelan v Jaslow}\textsuperscript{225} appear to have adopted without question the court’s finding that development of the structure and logic of the program generated a significant portion of the cost of producing a computer program.\textsuperscript{226} This justified the court’s protection of a program’s “structure, sequence and organisation”. By comparison, when the question was whether or not the user interface should be protected, the assertion was that a well-designed user interface was “extremely resource intensive”\textsuperscript{227}, and “typically the most time-consuming part of the

\textsuperscript{223} Architectural Works Copyright Protection Act 1990 104 Stat 5133.


development process." 228

This reasoning is analogous to "trivial acquisition" arguments. If a competitor is seen as able to obtain a portion of the market share of the original innovator through imitative competition, the reduced potential for monopoly profit fails to generate sufficient incentive to invest, inviting intellectual property intervention. Followed to a logical conclusion, some commentators would propose that all imitative competition be prohibited, at least for some appropriate degree of market lead time. For example, Samuelson et al. 229 propose a "cloning" prohibition for innovators to recoup research and development expenses. A major reason for this approach is the belief that reverse engineering – by writing new entirely new source code – is only "somewhat more demanding" than replicative copying of the machine code. Such an assumption is at odds with the analysis of software development in Chapter Five.

The problem with this economic argument is that it begins to suggest that abstract ideas, concepts, and other elements of what copyright would traditional treat as part of the public domain, should be protected. A common fallacy is to equate the commercial value of a software product with its function, and attempt to support this commercial value through


copyright through some prohibition on replication of the function or of
the "total concept and feel". Samuelson et al.\textsuperscript{230}, in using the "trivial
acquisition" argument, appear to confuse the means of creation with the
program behaviour and its functional design:

Such know-how is, inescapably, present on the face of the product,
and immediately evident on inspection. The hard-won insights
and innovations embodied in surface design are prominently
displayed by the program in operation ... Paperback needs nothing
more than the ability to run Lotus 1-2-3 and observe its behaviour
in order to clone it.

Such an argument inevitably leads to suggestions that "conceptual
metaphors"\textsuperscript{231} be protected through intellectual property laws. It is
difficult to see how a conceptual metaphor is something different from an
unprotectible "idea" in copyright doctrine.

Strong monopoly powers are seen as desirable for the stimulation of
levels of investment required for "revolutionary" invention.\textsuperscript{232} The
computer software industry is, however, widely accepted as incremental
and cumulative rather than revolutionary in this economic sense. The
price for an intellectual property monopoly over elements of surface
design, interfaces and total "concept and feel" is that identified by Landes
and Posner. Competitors are also "authors" who are discouraged from

\textsuperscript{94} p. 2308 at pp. 2337 - 2339.

\textsuperscript{230} Ibidem pp. 2333 - 2337, quotation at p. 2335.

\textsuperscript{231} Such as the concept of "virtual paper" in word processing.

\textsuperscript{232} Scherer, F.M. (1991) "Antitrust, Efficiency, and Progress" in First, H. et al. (eds)
Essays on Legal, Economic, and Political Policy p. 140.
creative investment by the barriers intellectual property law can erect.233 Broadening the scope of protection to ideas, metaphors, look, feel or functionality forecloses entire areas of potential output by other authors.

One element of the policy argument protecting surface design elements and interfaces is that software is a creative process analogous to traditional literary works. The following section challenges the proposition that software is creative in the literary sense.

6.3 Creativity arguments

Although some commentators have argued that:

The imagination, originality, and creativity involved in writing a program is comparable to that involved in more time-honoured literary works.234 such comments are misleading. The assertions relating to creativity are intended to demonstrate that computer programs should be protected in the same manner as the other works on the account that they involve a similar creative process. Whilst it is true to assert that all invention involves creativity, some inventions are more restrictive in the scope of their expression than others.


One reason for the assertions of substantial creativity is that jurisprudence, particularly in the United States, has emphasised that some minimum element of creativity\textsuperscript{235} – stemming from the requirement of "originality" – is necessary for copyright protection.\textsuperscript{236} This requirement was recently reinforced by the decision of the US Supreme Court in \textit{Feist v Rural Telephone}\textsuperscript{237}

Under copyright, protection of the artefact is not intended to limit access by other authors to the means of creation of further works. Although computer programs may appear to be similar to other works, in that respect, upon a deeper analysis and understanding of the process of programming and on the manner in which programs operate, it can be seen that the similarity is superficial at best. As a program is ultimately rendered in binary, if it is to be efficient it must be limited in its expression. For each particular program in each particular source language, there is an ideal set of algorithms which would result in an optimal solution in terms of the desired functionality and speed. The consequence of this is that good programming is exemplified by functionality, economy of code and efficiency in causing the computer to

\textsuperscript{235} L. Battlin & Sons Inc. v Snyder 536 F.2d 486, 490 (2d Cir. 1976).


Chapter Six – Implications for Copyright

carry out the programmer’s instructions.\textsuperscript{238} From this it can be seen that a program’s expression is necessarily directly related to and inseparable from the substantive outcome desired in terms of the program’s operation. Portraying programmers as having distinctive “styles” suggestive of an artistic activity akin to writing a novel\textsuperscript{239} marginalises the object of efficiency as a necessary part of programming and fails to recognise that the public domain for a programmer is far more limited and technology-dependent than it is for other authors. The nature of programs, especially the fact that their utility lies in interpretation by a machine rather than a human being, should have led to a deeper consideration of their suitability for protection under copyright.

It is the understandability of the symbolic language and use of English based mnemonics which has led “almost inexorably”\textsuperscript{240} to the characterisation of source code as “literary works”. The improper argument which then follows is that analogies to infringement principles in traditional literary works is appropriate. The case is markedly different for computer programs which interact with a processor and direct that

\begin{itemize}
\end{itemize}
processor to perform in certain ways. A computer program is the machine. Focus on the symbolic language used in programming is misleading. Samuelson et al.\textsuperscript{241} correctly described programs as “machines whose medium of construction is text”.

There is also a common thread in the generation of “time-honoured” works which is absent from computer programs. This common thread that can be drawn upon these works despite the constraints under which they laboured, is that these works have the power to evoke an emotional response. Although computer programs contain their own vocabularies, grammar and syntax, as do these other works, a program has no ability to convey emotion. A CPU is not alive and cannot care about the elegance of the program which it is running. As was demonstrated in Chapter Five, at some point before the running of a program, the program will check the words entered in source against a predefined vocabulary. This vocabulary usually exists in the form of a library which tells the program which locands within the machine to send the instruction to once the instruction exists in executable code. The result of this is that whilst words expressed in languages by which human beings communicate contain both connotation (meanings and emotional states evoked by the words) and denotation (literal meaning), programs have only denotation.


When a program is given the instruction "run", the CPU cannot ponder whether the program intended it to trot or move to a swift canter (it does neither).

Of all the alleged similarities with other works, computer programs most resemble architectural plans. Like computer programs, even fanciful architect's plans have effective constraints placed upon them by their functional nature. The design of a structure will depend on user requirements, subject to constraints such as the need for tensile strength, load bearing and planning laws. In terms of the nature of their creation, programs and architects plans are similar, however, in terms of their execution, this similarity wanes. A CPU must execute the program which is passed to it in executable form, whereas, builders have liberty to alter and interpret plans, and to use a range of materials to comply with a given plan. The quality of "inclusions" and finishing touches often have significant impact on the value of the completed building. Architectural plans have both connotation and denotation as their ultimate instrument of interpretation and implementation is human. Computer programs are incapable of expressing connotation as their ultimate instrument of interpretation is a CPU which can only understand limited mathematical algorithms.

It is questionable whether it is socially desirable to discourage
standardisation in programming architectures.\textsuperscript{242} Whereas variety has social value in drama and literature it has little value in programming, where standardisation is more advantageous.\textsuperscript{243} The need for efficiency and reliability within program construction has, in practice, produced a set of "standard" programming techniques, which constrain the design choice regarding substance, structure, and form of programs.\textsuperscript{244} One aspect of recent industry and academic thinking seen as the key to improving program development productivity and quality is the notion of reusability.\textsuperscript{245}

Many commercially available development environments include a prepackaged application framework, generating source and object code for commonly used routines such as maintaining the user interface, accessing mathematical functions, and manipulating strings of text. These frameworks break code into reusable "building blocks" to avoid unnecessary reinvention and encourage development as a process of component composition within an overall product design. One commercial advantage of software reuse is its potential for reducing the

\begin{thebibliography}{9}
\end{thebibliography}
cost of software development. Additionally, the reuse of robust components can increase the reliability of any final composite product, and enhance debugging activities.246

Reusability is also implicit within an object oriented programming (OOP) philosophy.247 The essence of OOP is that a program is built upon classes and objects. A class is a description of an entity used by a program. That entity may be a menu, an invoice, or a customer file. Classes describe these entities in terms of properties (data) and functions which can be applied to these properties. Once established, a class is used as a template to create objects (special instances of that class) which the program uses. These objects draw upon a class library whereby all objects created from the one class will share a minimum set of properties and functions. In turn classes can draw upon properties and functions of other classes through a process of inheritance. Languages such as C++, based on class libraries and inheritance, promote software development as a process of modifying existing code libraries and drawing upon base classes rather than writing entirely new code. New software projects are increasingly a combination of general classes already developed and objects - special instances of the classes - built particularly for the project. As more projects are completed by an entity, some of the classes developed for any


particular project can be incorporated within the general code library to be the basis for future projects. This suggests far greater emphasis on the design stages of any software project and larger up front costs in creating the reusable code library.

There is also an increasing trend towards programming within systems software environments which use sophisticated graphical user interfaces (such as MacOS or Windows). Many programmers will develop or purchase application frameworks which provide the "shell" or operating infrastructure for their software. The usefulness of these frameworks is in recognising common needs for program interaction with the systems software and providing customisable solutions to these needs. Extensive source code is provided which is elaborated by the programmer. One consequence of the use of such application frameworks is that increasingly, programs will exhibit similar behaviour at the user interface level. Two examples of such frameworks within the MacOS environment are EasyApp (written in C) and PowerPlant (written in C++). Both are outlined in Appendix Four.

A recurring metaphor is that programming is a engineering process, not an artistic endeavour. Writing programs is therefore an industrial

---

design process akin to the design of physical machines. Davidson\textsuperscript{249} agreed that a major fallacy in the literature of software protection is to view it as the result of artistic creativity. He argued that software is an engineering, not an artistic process:

Software is created like most engineering products: by a process of problem or project definition, followed by designing the product (the program), creating a prototype (writing the source code), testing the prototype (debugging), and ultimately realizing a commercially marketable product.

Computer programs do not exhibit creativity in the same way as other literary and artistic works. The implication for policy is that the extension of intellectual property protection, via the “Cheshire cat grin” fallacy, to justify literary tests for infringement such as “non-literal” copying, is improper. Where a competitor can infringe by imitative copying even where there is no replication of source or object code, this has serious consequences for incremental development and the realisation of network externalities. The major focus of this conflict is discussed in the following section, which examines the question of protection of elements of compatibility and interfaces.

6.4 Compatibility and Interfaces

Compatibility is about ease of communication, both between systems and its users, and between parts of the system itself. The former can be

Chapter Six – Implications for Copyright

generally described as the “user interface” and the latter as “internal interfaces”. Examples of internal interfaces would be between a particular hardware (chip) architecture and an operating system, or between an operating system and an application. Interfaces will also exist between systems and other systems. Communications protocols, many of which are defined by market leaders, are increasingly important as means for transmitting information and enabling access to networks.\textsuperscript{250} The question of compatibility raises serious economic consequences in situations where strong network externalities exist.\textsuperscript{251} Elements necessary to achieve compatibility, including features of a user interface which is a de facto standard in the marketplace is seen by some commentators as a significantly different type of intellectual property question.\textsuperscript{252}

One conclusion from the existence of network externalities\textsuperscript{253} is the demonstration of the importance of standardisation in some elements of computer programming. From an informational perspective, the potential restrictions on the realisation of these externalities – essentially information flows – is of serious concern. This section critically examines


\textsuperscript{253} Discussed in Chapter Three.
the arguments promoting the copyright protection of elements which are surface design, standards, user interfaces, or otherwise related to compatibility.

6.4.1 Economic justification

The major justification for protection of surface design and compatibility elements rests on incorrect economic propositions already discussed in section 6.2. According to traditional analysis, the fact that the "look and feel" of software is considered commercially valuable, or is expensive to create, warrants monopoly protection per se. Some studies have shown that development of a sophisticated graphical user interface (GUI) can be the subject of an expensive processes of "human factors engineering" involving elements of cognitive psychology. Economic commentators appear to assume that where expenditures are necessary to create a standard, and its existence is valuable, that the copyright incentive is justified. It seems to follow that the policy objective of copyright law is to protect the valuable elements of a computer program, which is widely perceived to be partly driven by the popularity of its interface.


Kellner\textsuperscript{257} described investment in user interface construction as building "consumer recognition" of programs.

Computer program developers have great financial incentive to restrict the ability of competitors to adopt similar screen displays and user interfaces.\textsuperscript{258} Developers have argued that without legal protection, any competitive advantage is short lived and object to competitors imitating features which have become popular in the marketplace.\textsuperscript{259} This debate is also characterised by a reliance property rights rhetoric. McGrath\textsuperscript{260}, for example, argued that denying protection to de facto interface standards would "divest" copyright owners of their property rights.

\textbf{6.4.2 Standards competition}

One traditional incentive argument is typically extended to the picture of firms setting out to develop new and different standards for the marketplace. Some commentators argue that competition amongst interface standards is desirable and that legal protection is warranted to

\begin{thebibliography}{99}


\end{thebibliography}

\begin{flushright}
\textsuperscript{page 143}
\end{flushright}
provide such an incentive. The outcome of such a competition is the emergence of a theoretically superior standard. This is known as the "QWERTY" argument, by analogy to the fact that standardisation of the QWERTY keyboard has precluded widespread adoption of the superior Dvorak configuration. The theme of not being "inefficiently locked into old choices" is evident in policy discussion of copyright and computer software. Landes and Posner's assertion that the creation of an industry standard as a "tribute to the expressive skills of the particular manufacturer" belies an assumption that the market will invariably choose the qualitatively superior standard.

Such an argument may be misplaced since network externalities may entrench an arbitrary standard. Some firms may adopt pricing and


marketing strategies to ensure that their interface becomes popular. The forces of network externalities can make the introduction of new and competing interfaces costly and subject to a high risk of failure, even though the interface may be theoretically superior. Menell argued that the availability of legal protection for interface standards may provide an incentive for firms which a large base of installed users to adopt incompatible interface standards to reap increased monopoly rewards. In commenting on the success of Microsoft one industry analyst wrote:

The success of any technology company depends to a great degree on its ability to forge new technology standards and persuade the rest of industry to adopt them. Gates’ flair for doing that has made him the envy of one of the world’s most lucrative and strategic businesses.

6.4.3 Standardisation
The arguments set out above ignore the effect of network externalities, and make the implicit assumption that variety is just as important in user interfaces as it is in the content of novels and motion pictures. Paley, for example, argued that standardisation “in its own right stifles creativity”. Although Landes and Posner speak of copyright promoting

---


economic efficiency through maximising "the benefits from creating additional works."\textsuperscript{272} There is an implicit assumption that benefits necessarily follow in linear fashion from the number of works produced. This assumption does not necessarily hold for some elements of software artefacts. While creative diversity and originality may be a virtue in literature and the arts, the technology literature stresses consistency, intuitiveness, reusability and interoperability as features of a desirable information infrastructure. Efficiency and compatibility are objectives in computer programming, yet copyright is not designed to achieve uniformity, but rather variety.\textsuperscript{273}

In the software industry, Menell\textsuperscript{274} noted that major firms work together to develop operating systems and interfaces in the public domain. This behaviour assists in defining markets and directing development resources to improvements of products within these particular standards. Besen and Raskind\textsuperscript{275} noted that users benefit from the existence of standardised interfaces through a larger array of complementary inputs.


\textsuperscript{273} Goldstein, P. (1986) "Infringement of Copyright in Computer Programs" University of Pittsburgh Law Review, Vol. 47, p. 1119. In some cases, however, it has also been shown that compatibility standardisation will create "mix-and-match" opportunities, thus increasing variety: Matutes and Rigabeau (1988) "Mix and Match: Product Compatibility Without Network Externalities" Rand Journal of Economics, Vol. 19, p. 221.


\textsuperscript{275} Besen, S.M. and Raskind, L.J. (1991) "An Introduction to the Law and Economics of
available, and the reduction in switching costs between systems or applications.276 The shape of the user interface is increasingly being determined by empirical research into human factors.277 This has influenced choices as to command based, menu-driven and natural language programs,278 menu design,279 and colour and highlighting techniques.280 Beutel suggests that protection would be most appropriate were an approach adopted which would not extend protection to any aspects of a program dictated by logical structure or efficient design requirements, including specific operating characteristics, standard algorithms and features dictated by hardware and program requirements.281 Standardisation of user interfaces also prevents "lock-
In” or the imposition of switching costs. Permitting compatibility encourages entry, a broader range of choice and more competitive prices. Samuelson et al. stated:

Consumers are also becoming less tolerant of stand-alone, idiosyncratic software. Hence, the market is increasingly driven by the need for interoperability, in several senses: Software must operate on a variety of hardware platforms, across networks, and most of all, programs must work together.

Farrell noted that the conventional assumptions regarding intellectual property protection no longer hold when confronted by network externalities. Strong protection may provide an inappropriate incentive for an innovator to encourage widespread adoption, through penetration pricing or other means, of their innovation even when it may not be socially useful. Innovators who develop potentially superior innovations may derive no reward at all, or may harm the base of installed users who choose not to adopt the innovation – a “stranding” externality. Farrell argued that within network externality industries, the goal of intellectual property protection ought to be the encouragement of useful innovations while also ensuring the network benefits are fully realised. He concluded,


Chapter Six - Implications for Copyright

in relation to computer software that:\footnote{Ibidem p. 47.}

[Useful Innovations should have some degree of protection, but
the user interface, the format for data storage and transmission, and
other relatively arbitrary aspects that must be standardized in order
to achieve compatibility benefits, should be unprotected, so that
other software developers are encouraged to achieve
standardization.

Public Benefit” Rutgers Computer and Technology Law Journal, Vol. 14, p. 1 at p. 62.} argued that the judicial applications of copyright to the user
interface will protect design ideas underlying computer programs, with
Interface Designers on the Software Copyright ‘Look and Feel’ Lawsuits” Jurimetrics
Journal, Vol. 30, p. 121.} found that
79% of user interface designers surveyed did not believe copyright
protection was appropriate for elements of the interface which were de
facto standards.

6.4.4 Implications
A number of interrelated policy recommendations arise from the above
analysis, and have been articulated by some commentators already
mentioned.

At a basic level, it can be seen as an improper use of intellectual property
to permit dominant firms to protect their market position by the
imposition of switching costs by proprietary ownership of particular

\footnote{Ibidem p. 47.}


Interface Designers on the Software Copyright ‘Look and Feel’ Lawsuits” Jurimetrics
Journal, Vol. 30, p. 121.}
interface standards. Hemnes\textsuperscript{288} argued that the law should not protect "any part of a computer program that potential buyers would prefer not to re-learn before buying a competitive product".

It is the user interface which is extremely critical to the realisation of network externalities. Menell\textsuperscript{289} argued that little protection should be given to user interface elements of a computer program.

A broader set of policy recommendations would agree with Teter,\textsuperscript{290} who argued that copyright should not protect the following elements of software:

- elements dictated by efficiency;
- internal interface elements required to achieve compatibility; and
- elements of user interfaces which have become de facto standards.

As a general statement is clear from the discussion in this chapter and the model developed in Chapter Five that broadening of copyright protection of computer programs to protect "non-literal" copying is undesirable from the perspective of technologication innovation. This is where "non-literal" copying means imitating some aspect of behaviour of another


program without access to the source code of the original. In the presence of network externalities, a prohibition against "non-literal" copying can deliver profound market power to a firm able to influence industry standards.

6.5 Summary

This Chapter has examined the implications for computer program copyright of the model of technological innovation developed in Chapters Four and Five. Key to this discussion is the proposition that realisation of network externalities and avoidance of potential abuses of market power were desirable for copyright policy. Economic arguments which assume that innovation is driven in linear fashion by the degree of monopoly and the size of the commercial return to innovators ignores an important contrary factor. This creates a bias in the economic justification which, together with assertions about artistic creativity, leads inexorably to policy conclusions about the desirability of preventing imitative competition, and treating interfaces as proprietary. Such a policy has profound implications for the realisation of network externalities in the software industry.

As was discussed in Chapter Five, much of the policy literature has misconceptions of the nature of programming and the degree of access to the knowledge underlying the creation of programs distributed only in machine code. This thesis rejects this reasoning. The correct model of innovation should be that of reverse engineering. The source code of an
innovator is effectively confidential information. Competitors must engage in their own learning and engineering process, which is both costly and time-consuming. This reasoning applies just as strongly to the question of interfaces as it does entire programs. Any competitor wishing to adopt a particular interface feature will need to implement their own programming solution. As Spector\textsuperscript{291} stated:

Protecting the code that implements the interface should be sufficient to provide reward to the interface designer.

Based on the discussion in Chapters Five and Six, the following conclusions can be reached in respect of intellectual property protection of software:

• replication of machine code ("piracy") should be prohibited, to ensure that the value of the IT artefact is appropriable to the innovator;

• the law should encourage an innovation model based on reverse engineering, on the basis that source code (codified knowledge) remains effectively confidential to the innovator, and competitors must implement their own source code solutions;

• the law should not prohibit, but encourage, imitative competition provided that competitors write their own source code;

Chapter Six – Implications for Copyright

- decompilation should be permitted to provide a limited degree of access to some subsidiary elements of programs, such as interfaces between operating system and application, and communications protocols;
- programming elements dictated by requirements for efficiency and compatibility should not be the subject of intellectual property; and
- reproduction of elements of a user interface should not be grounds for infringement.

The following chapter examines the current legislative and judicial approaches to computer program copyright in Australian and international jurisdictions. These approaches will be critically examined against the policy recommendations above. The intention is to assess the degree to which current interpretations of copyright laws in respect of computer programs promote or detract from the informational economic perspective set out in the earlier chapters of this thesis.
Chapter Seven
Evaluation of International Copyright Laws

7.1 Introduction

The purpose of this chapter is to assess the current state of copyright legislation and case law internationally, with specific emphasis on the United States, Europe and Australia. The operation of the law will be compared with the policy objectives outlined in the previous chapter. Principally this thesis agrees with Karjala\(^{292}\) who stated that the software protection goal should be protection against "piracy", meaning literal or near-literal copying of program code. The role of copyright is to prohibit misappropriation in this sense.

As discussed in the previous chapter there are cases and a supporting literature which would extend protection beyond program code to include more abstract "non-literal" elements. Closer examination of the reasoning behind examination of the "look and feel" and "structure, sequence and organisation" cases reveal that the approaches of the courts can be reconciled with the policy objectives developed in this thesis, provided that a distinction is drawn between misappropriating source code and developing a competing product writing fresh source code. Such

\(^{292}\) Karjala, D.S. (1994) "Recent United States and International Development in
a distinction would assist courts in reconciling evidentiary and doctrinal issues currently confused within the case law.

The focus of the chapter is on the effectiveness of two key mechanisms for maintaining the balance between providing a basic level of protection while encouraging innovation, already mentioned in section 4.4.3 First is the “idea-expression” distinction, drawing the line between the protected and the unprotectable, permeates the case law and associated literature but has been applied with little consistency. The second is the doctrine of “fair use” (or “fair dealing”), which authorises acts of reproduction and adaptation of works (which would otherwise infringe) for particular purposes such as research and study. Of particular importance to computer programs is whether fair use endorses a reverse engineering approach generally, and authorises disassembly (decompilation) in particular. A central concern is the degree to which interfaces, both internal interfaces (with hardware or software) and user interfaces, may be imitated by competitors.

The jurisdictions were selected as representative of distinct perspectives in terms of copyright policy. The United States as the largest domestic market and dominant exporter of software internationally, represents a key legal environment and the overwhelming proportion of litigation concerning software copyright issues has taken place there. Europe with a growing telecommunications and information technology infrastructure

is an important potential competitor to the United States. Australia by contrast is a small country and net importer of software, although with very similar legal traditions to the United States in the area of intellectual property. As discussed below, the different circumstances and legal-political environment of the European Community (EC) have created an explicit statement of software copyright policy lacking in the other jurisdictions, with some notable endorsements of reverse engineering and compatibility.

The conclusion is that while the current direction of the law (as at December 1997) appears to be closer to the policy recommended by this thesis, this may be short lived.

7.2 Legislation

At a basic level the source code and object code of computer programs is protected under copyright as a “literary work”, in each major jurisdiction achieved by specific legislation.

7.2.1 United States

The United States Congress created the National Commission on New Technology Uses of Copyrighted Works (CONTU) to report on the question of software copyright. The recommendations of this committee were embodied in 1980 amendments (Computer Software Copyright Act 1980) to the 1976 Copyright Act. Congress adopted a definition of a
computer program as:293

A set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.

Under this definition copyright may protect both readable source code as well as human imperceptible machine code (object code). In the United States, this proposition was confirmed by Apple Computer Inc. v Franklin Computer Corp294.

7.2.2 Europe

In 1985, the Commission of the European Community (EC) initiated discussion on harmonisation of national laws on intellectual property.295 In 1991, the EC's Council of Ministers approved a directive requiring European copyright protection of computer software (EC Directive).296

Article 1 of the EC Directive required member states to protect computer programs as literary works under copyright. The United Kingdom has earlier incorporated computer programs with the passing of the Copyright (Computer Software) Amendment Act 1985 as a temporary measure, prior to a review of intellectual property and the later passing of the Copyright,


294 714 F.2d 1240 (3d Cir. 1983); 104 Sc. Ct. 690 (1984); see also Apple Computer Inc. v Formula Int'l Inc. 725 F.2d 521 (9th Cir. 1984) in respect of systems software.


Chapter Seven – Evaluation of International Copyright Laws

Designs and Patents Act 1988. This legislation was in turn amended to bring it into line with the EC Directive in 1992. 297

Article 1 did not, however, contain any definition of a "program", although an earlier Explanatory Memorandum suggested "a series of instructions which causes a machine to perform its function". The EC Directive does include "preparatory design materials" as part of a program.

7.2.3 Australia

In Australia, the Commonwealth passed special legislation to incorporate computer programs within the Copyright Act 1968. Computer programs are "literary works" 298 and computer program defined in the following terms:

"Computer program" means an expression, in any language, code or notation, of a set of instructions (whether with or without related information) intended, either directly or after either or both of the following:

(a) conversion to another language code or notation;
(b) reproduction in a different material form,

to cause a device having digital information processing capabilities to perform a particular function.

Part of the explanation for the detailed definition within the Australian


298 Section 10(1).
legislation is that it was legislated in response to a court finding that computer program machine code was not necessarily protected under traditional principles of copyright: *Apple Computer Inc. v Computer Edge.*299 The definition was based in part on the World Intellectual Property Organisation’s (WIPO) model provisions for protection of computer software and a meeting held in Canberra during 1984 with the Attorney-General’s Department.300 The High Court later confirmed the protection of machine code under the 1984 amendments.301 In 1995 the Copyright Law Review Committee (CLRC) recommended that the definition of “computer program” in the Act be replaced by a broader definition effectively replicating that in the United States.302

### 7.3 The idea-expression distinction

A key approach to the determination of questions of copyright within information technology is the application of the idea-expression distinction. Copyright does not protect ideas, schemes, systems or methods but is confined to the expression of those ideas.303 This principle, at least in abstract, is agreed by all three jurisdictions. Explicitly

---


303 *Hollinrake v Trustwell* [1894] 3 Ch 420 at 427.
within the United States legislation, copyright does not extend to any “idea, process, system [or] method of operation”.\textsuperscript{304} Although no similar statutory provision exists in Australia, it has been held that the United States legislation merely reflects the general law principles.\textsuperscript{305}

Under the EC Directive Article 1:

Ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright under this Directive.

The application of the idea-expression dichotomy to software has often been based on the approach of the United States Supreme Court in \textit{Baker v Selden},\textsuperscript{306} considered to be the landmark ruling on this issue. In that case, a book published by the plaintiff presented a system for bookkeeping. Included in the book were example ledgers which demonstrated the system. The defendant did not copy the book but instead implemented the system of bookkeeping. The Supreme Court found there to be no infringement, on the basis that the description of the system did not give the author an exclusive right to use of that system. Only the expression of the system, its articulation within the book, was protected. Applying \textit{Baker v Selden} to computer programs, the source code and object code of the program would constitute a unique “expression” deserving of protection. The functions performed by the program would be an

\textsuperscript{304} 17 U.S.C. § 102(b) (1988).

\textsuperscript{305} \textit{Powerflex Services Pty Ltd v Data Access Corporation} [1997] FCA 490.

\textsuperscript{306} 101 US 99 S Ct (1880).
unprotectable "idea". Three potential methodologies for applying the distinction are discussed below, each emerging from United States courts. The first arises from a situation of imitative competition, in that the defendant wrote their own program without access to the plaintiff's source code. The second and third both deal with situations of misappropriation, where there was a degree of access to protected source code. This difference in facts between the cases will assist in reconciling their approaches in section 7.4, based on an confusion of doctrinal with evidentiary questions.

7.3.1 Synercom

A pioneer case suggesting that interfaces could be imitated in competing programs to promote compatibility was *Synercom Technology Inc. v University Computing Co.*307 which dealt with a structural engineering program. At issue was whether the defendants had infringed input formats used in the Synercom program. These formats were a number of cards with lines and spaces arranged in a particular order indicating how the engineering data should be entered into the computer. The intention was that the users would be able to write the necessary data on these cards to facilitate their use of the program. The court found that the blank cards to be protected literary works for the purposes of copyright. The defendants did not reproduce these cards but their competing program's

---

input routine was structured on the particular sequence of data in the Synercom cards. Users of the defendant's program would then be able, without retraining, to enter data already written on Synercom cards to make similar engineering calculations. The defendant's manual provided detailed instructions on data entry which the court found to be a "mirror image" of the Synercom forms.

The court found that substantial similarity in the two descriptions of the data entry process, the logic and sequence of the process adopted by Synercom, was an unprotectable idea which the defendants were free to imitate in their program. Any similarity in expression was inevitable results of borrowing the unprotected ideas. The court used the analogy of the figure-H pattern of a gearstick. Although the manual for the original vehicle using the gearstick would be protected by copyright, a second manufacturer adopting the unprotected figure-H pattern would be at liberty to describe its use despite any similarity this may have to the original description.308

The Synercom case is significant in that its clear implication is that achieving compatibility is not of itself prohibited by copyright. It is also important in terms of the model of software innovation developed in the previous chapters. The court explicitly endorsed the "borrowing of an idea" in terms of development of a competing program imitating

308 Ibidem p. 1013.
behaviour of the first but without access to or copying from the source code. The court stated:

Hence [the defendant’s] preparation of a FORTRAN preprocessor program from the descriptions contained in the manuals cannot constitute an infringing use provided this was done without copying of the plaintiff’s FORTRAN program, as it was.

This thesis endorses the Synercom approach as consistent with the objective of promotion of innovation from an information perspective. The case has, however, been overshadowed by rulings that, despite no objective similarity in code, similarities in “non-literal” elements can constitute infringement. These cases are discussed below.

7.3.2 Whelan

A significant decision in terms of subsequent case law in the United States was Whelan Associates Inc. v Jaslow Dental Laboratory Inc. ("Whelan"), a decision of the Third Circuit. Whelan wrote a program in the language EDL for managing the business operations of Jaslow’s dental laboratory, called Dentalab. Whelan was to own copyright in the program while Jaslow was to sell it to others on behalf of Whelan. Two years following the completion of the program, Jaslow began selling its own dental management program, Dentcom, written in BASIC but performing similar functions to the program written by Whelan. Experts at trial found little similarity in source code (principally due to the different

---

309 Id.
languages used) and distinct logic in the sub-routines. The experts agreed, however, that there were "overall structural similarities" in that five modules had similar functions and there were noticeable similarities in file structures and screen displays.

The issue on appeal was whether the observed similarities in structure were sufficient to ground an infringement of Whelan's copyright. This turned on the question of whether the structure of a program is part of the idea or its expression. Whelan held that "expressive" non-literal elements should be afforded protection provided that they were not merged with their underlying ideas. The merger doctrine denies protection to expression where there were a limited number of ways in which to express a particular idea. The policy behind merger was that by protecting expression, the law did not provide a monopoly over the underlying idea. Using this as the basis for applying copyright to the computer program, the court held that the "idea" of the program could be found in its "function or purpose" which it held to be the efficient management of a dental laboratory. The defendant had used similar subroutine structures, file structures, and screen displays. Since a range of these structures and displays could support the same "idea" (purpose), the particular "structure, sequence and organisation" used by the plaintiff was

---

310 797 F.2d 1222 (3d Cir. 1986).
311 e.g. Morrissey v Proctor & Gamble Co. 379 F.2d 675 (1st Cir. 1967).
312 797 F.2d at pp. 1238-1240.
protected by copyright. The test stated by the Third Circuit was that:

\[313\] ... the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea.

The case has suffered heavy criticism although its impact on subsequent case law was considerable. Gross\[314\] argued that the court's approach in Whelan begged the question since in particular it did not explain the methodology for arriving at the particular "idea". The level of abstraction of the idea in Whelan was at a very high level – the function of the program. In addition, this function was delineated in terms of overall purpose rather than the specific functions performed during its operations. It is arguable that this overall purpose is so abstract as to render the idea-expression distinction meaningless.\[315\] Drahos described the approach of Whelan in the following terms:\[316\]

This approach basically amounted to increasing the abstractness of the abstract object so that identity judgments concerning computer software could include their non-literal elements.

Englund\[317\] argued that while easy to apply, the rule in Whelan would

\[313\] 797 F.2d at p. 1236; it can be noted that this approach reflected the reasons in Apple Computer Inc. v Franklin Computer Corp. 714 F.2d 1240 at p. 1253 (3d Cir. 1983).


\[317\] Englund, S.R. (1990) "Idea, Process or Protected Expression? Determining the Scope of
have undesirable consequences. Ideas for program functions and structures typically arise from observing the works of others and is then improved upon and combined with original material. Prohibitions on programmers imitating behaviour at any abstraction below an overall purpose would effectively provide a monopoly over an unprotectable process, and hamper the innovation process within the industry.\textsuperscript{318} 

Whelan entrenches market leaders and may consequently encourage software inefficiencies.\textsuperscript{319} From the perspective developed in this thesis, acceptance of a Whelan approach would be harmful to future innovation within the software industry.

### 7.3.3 Computer Associations v Altai

The landmark case which articulated an alternative methodology to Whelan was \textit{Computer Associates International Inc. v Altai Inc.}\textsuperscript{320}. In that case, Computer Associates owned CA-SCHEDULER, a program which controlled the operation of a computer in carrying out a schedule of tasks. Integrated within CA-SCHEDULER was a compatibility component, called

\begin{itemize}
\item 982 F.2d 693 (2d Cir. 1992).
\end{itemize}
ADAPTER which translated the scheduler language into machine language for execution. The purpose of ADAPTER was that CA-SCHEDULER could be used with a range of systems software and hardware architectures. Altai began marketing ZEKE, a competing scheduling program which unlike CA-SCHEDULER was only able to run on one operating system. An Altai employee who had access to the ADAPTER source code in violation of his previous employment with Computer Associates, wrote OSCAR which, in version 3.4, was released with ZEKE by Altai to operate ZEKE on multiple systems. Computer Associates sued for infringement and Altai eventually admitted to copying 30% of ADAPTER for OSCAR 3.4. Altai subsequently rewrote OSCAR by way of a cleanroom process to purge it of all code copied or modified from ADAPTER and then released OSCAR version 3.5. The Second Circuit held that OSCAR 3.5 did not infringe Computer Associates’ copyright in ADAPTER.

The court explicitly rejected *Whelan*, noting that the major flaw in that case was the assumption that only one “idea” underlies a given computer program. *Altai* also rejected the proposition that the “structure, sequence and organisation” of a program was copyrightable expression. Instead the Second Circuit used a three stage procedure to determine infringement.\(^\text{321}\)

This has been described by commentators as a process of “abstraction-
filtration-comparison". Under this approach, the court first breaks down the original program into its constituent structural parts abstracted from the literal code. By examining each of these parts for:

- incorporated ideas;
- expression necessarily incidental to those ideas;
- elements dictated by efficiency;
- elements dictated by external factors; and
- elements taken from the public domain;

a court filters out all the non-protectable material, leaving a "kernel" or "golden nugget" of creative expression. Finally the court compares the remaining expression and determines whether the protectable elements of both programs are substantially similar as to find infringement.

Effross noted that Altai's exclusion of elements dictated by efficiency paralleled the copyright doctrine of merger, while the elimination of element dictated by external factors was analogous to the scènes à faire doctrine. Similarities between programs based on efficiency might be attributed to the industry-wide pursuit of efficiency, not of copying, just as similarities reflecting standard industry practices or interoperability do

---


not necessarily represent copying. According to Altai the external factors giving rise to similarities would include:324

(1) the mechanical specifications of the computer on which a particular program is intended to be run;
(2) compatibility requirements of other programs with which a program is designed to operate in conjunction;
(3) computer manufacturers' design standards;
(4) demands of the industry being serviced; and
(5) widely accepted programming practices within the computer industry.

More importantly as a question of policy the court rejected the argument that programmers needed to be provided with extensive protection to invest in software development. The interest of the copyright law was not simply to confer a monopoly on industrious persons, but to advance public welfare by promoting the free use and development of non-protectable ideas and processes. The court observed that:325

... serious students of the industry have been highly critical of the sweeping scope of copyright protection engendered by the Whelan rule, in that it enables first comers to "lock up" basic programming techniques as implemented in programs.

At one level Altai suggested an evolving jurisprudence based on careful and pragmatic evaluation of alleged similarities, taking into consideration industry practices and real-world functional constraints.326 One

324 982 F.2d at pp. 909-910.
325 982 F.2d 693 at p. 708.
implication for software development was as an encouragement for competitors to imitate features of best-selling computer programs.\textsuperscript{327} This approach, closer to the policy in this thesis, was reflected in some decisions heard in the same year. For example, in \textit{Apple Computer v Microsoft Corp.}\textsuperscript{328} the court rejected Apple's argument that the overall look and feel of its Macintosh graphical user interface was infringed by the Microsoft Windows. It examined instead each of the distinct features surrounding the systems software functions, and suggested that items such as the "trash can" icon and the "zooming rectangles" feature when opening a document or application might be protectable, but in each case were not protected due to a limiting doctrine such as \textit{scènes à faire} or merger. The court also cited \textit{Synercom} with approval, drawing an analogy between the visual displays and user commands of the Macintosh user interface and a car's "dashboard, steering wheel, gear shift, brakes, clutch and accelerator".\textsuperscript{329}

Using the same approach, \textit{Brown Bag Software v Symantec Corp.}\textsuperscript{330} found that Symantec's "Grandview" outlining program did not infringe

\begin{flushleft}
\textit{Programs: Computer Associates v Altai and Other Recent Case Developments" European Intellectual Property Review, p. 351.}
\end{flushleft}


\textsuperscript{328} 799 F.2d 1007 (N.D. Cal. 1992).

\textsuperscript{329} \textit{Ibidem} at p. 1023.

\textsuperscript{330} 960 F.2d 1465 (9th Cir. 1992).
Brown Bag Software’s “PC-Outline” program. The alleged similarities were divided into four major features each of which was found not to be infringing. First was the list of options in the opening menu, found to be unprotectable concepts fundamental to a host of computer programs, such as opening files, editing and printing. Second was a set of nine functions in the menu bar, held to be incidental to the idea of a computer outlining program. Third, the use of pull-down windows was held non-original and standard practice within the computer industry, and finally, the use of a blue background which was held to be merely functional. On appeal, the Ninth Circuit endorsed the “analytic dissection” of the two programs and stated that individualised evaluations to determine whether similarities result from unprotectable ideas or protectable expression were required.

A difficulty is that the Altai approach, however, is that it leaves open the possibility that subsequent courts may still choose to limit or expand the “kernel” of protectable expression. One example was CMAX/Cleveland Inc. v UCR Inc.\textsuperscript{331} which appeared to rule that program design is a protectable non-literal element.

In applying the Altai procedure, the court found that file structures (the selection and arrangement of field definitions within a file) were protected on the basis that being arbitrary, not alphabetic or systematic,

they were not dictated by efficiency or industry considerations. Using the same reasoning, transaction codes used by the program as labels for shortcuts to executing particular transactions were protected. According to the court, having to retrain employees to learn new transaction codes in a competing program did not constitute an external factor so as to deny copyright. Another danger with the Altai approach is that it leaves open the possibility of copyrightable expression being conferred upon non-literal elements which are genuinely new and unusual, falling outside the filtration process as outside industry practice or not dictated by external factors. Effectively the court would be conferring patent-type protection on useful functions or user interface elements which meet an implicit "novel and non-obvious" threshold. There is also the danger of a chicken-and-egg problem. In a filtration test elements which have become standard practice are removed from consideration. At some point in time, however, all standard programming devices would have been seen as unusual or experimental, or perhaps unique to a particular application.

While superior to Whelan from a policy perspective, the approach in Altai remains at odds with the preferred approach in Synercom. The application of Altai to cases where the defendant had no access to the source code of the plaintiff may also lead to overly broad protection. The Lotus litigation discussed in section 7.6.1 is one example. A recognition that "non-literal" copying is driven by evidence rather than doctrine, discussed in the following section, would reconcile these approaches.
7.4 Non-literal copying

Since direct evidence of copying by admission or otherwise is typically unavailable in copyright infringement cases, the courts have developed a jurisprudence based on evidence of substantial similarity. In the United States the test of substantial similarity is typically expressed in terms such as the following:332

\[
\ldots \text{whether the accused work is so similar to the plaintiff's work that an ordinary reasonable person would conclude that the defendant unlawfully appropriated the plaintiff's protectable expression.}
\]

Given that substantial similarity turns on an inference from circumstantial evidence, the courts have adopted tests based on an overall pattern of similarity without regard for differences in detail. This is based on the need for the courts to ensure that "colourable alterations" to disguise copying do not permit infringers to avoid judgment against them. An infringing work therefore need only have captured, for example, the "total concept and feel" of the original.333 In applying this approach to computer programs, the courts have encountered significant difficulties. One of the difficulties is that legislators have provided little guidance to the courts in making policy choices in the approach to questions of infringement and determination of what is copyrightable.

332 Atari Inc. v North American Philips Consumer Elecs. Corp. 672 F.2d 607 at 614 (7th Cir. 1982).

333 Sid & Marty Krofft Television Prods. Inc. v McDonald's Corp. 562 F.2d 1157 at 1167 (9th Cir. 1977) quoting Roth Greeting Cards v United Card Co. 429 F.2d 1106 at 1110 (9th Cir. 1970).
Samuelson\textsuperscript{334} noted that CONTU, for example, displayed a remarkable lack of technological understanding. As McGahn\textsuperscript{335} stated:

The result is a variety of judicially created tests, which rely on amorphous terms such as "ordinary observer", "extrinsic/intrinsic", "total concept and feel" and "iterative".

The approach to infringement based on substantial similarity shares a great deal with the orthodox economic concepts presented in Chapter Two and stands in contrast to the information economics approach in Chapter Three. Under established copyright doctrine a plagiarist cannot use as a defence an argument as to the level of independent contribution, even if this is overwhelming. For example, in \textit{Meredith Corp. v Harper & Row Publishers Inc.}\textsuperscript{336} the defendant was a publisher who intended to create a book to compete with a leading book in psychology. One group of the defendant’s employees studied the book and provided a detailed report to a second group who had no access to the book. This second group then wrote the competing book. The court found similarity in structure and evidence to support approximately 11\% close similarity in the text, but that most of the remainder constituted new ideas, research, topics and some differing structure. Despite this originality and lack of access by the writing team to the original, the court found infringement. The


judgment is typical in suggesting that using an existing work as a model to create a new competing work may be prohibited by copyright. Once an inference of copying is supported by substantial similarity, the defence is usually limited to demonstrating that the similarity is merely coincidental. This jurisprudence makes an implicit assumption that innovation is by creation of wholly independent and original works. While this may reflect actual practice in the fields of art and literature, where the encouragement of diversity may be appropriate, its application to technological progress may be misplaced. Indeed, Karjala argued that substantial similarity was not appropriate for incremental developments since each new product is, it is unimproved portion, substantially similar to the first product.

It is a mistake, albeit promoted by some commentators, to argue that since the user is only interested in the “non-literal” elements of a program – its functionality and user interface – that the law should also be narrowly focussed on non-literal similarity. Instead the courts need to be aware that in dealing with program copyright they may be dealing with one of two distinct situations. The first is where the alleged copier had

338 See Chapter Six.
access to the original source code, such as where an ex-employee sets up a competing business using the source code of the original program as a short cut to software development. Making use of the source code of an existing program by inspection, determining its flow and structure, or plagiarising key routines can be seen as inappropriate. The second is where a competitor simply studies the program in its executable form and writes their own code to imitate some of the functions or interfaces. This behaviour by contrast can be seen as desirable to promote competition and cumulative innovation.

It is a trivial matter to change comments, names of routines, variables and other symbolic labels, as well as the order and structure of the source code files used in a software project. Given this opportunity for cosmetic modification of source code to disguise copying, similarities in structure, file formats, screen displays and other elements of the user interface do provide circumstantial evidence of actual copying of source code. The danger is that, in guarding against colourable imitation, the courts have created the jurisprudential basis for broad protection of non-literal elements, such as the "structure, sequence and organisation" in Whelan. The unstated assumption in these cases is that the alleged infringer is likely to have had access to the source code. Such an assumption may be entirely unrealistic in other cases.

The rationale for protecting "structure, sequence and organisation" in

cases like Whelan should be seen as based on evidence, not on copyright doctrine. Indeed, the courts will typically only be able to compare program “structure, sequence and organisation” in a source code form. One source code will be compared with another source code. It is clear than in Whelan, Altai, and cases which followed the courts were principally comparing two sets of source code as the basis for its decision. In Plains Cotton Co-op. Assn. v Goodpasture Computer Serv. the same programmers wrote both programs and had access to the designs of the first when writing the second. Similarly, in SAS Inst. Inc. v S & H Computer Sys. Inc. the defendant held a licence for much of the plaintiff’s source code.

Conley and Bryan argued that there is no a priori basis for treating the test of substantial similarity as the sole determinant of liability where more direct evidence is available. Substantial similarity is at best an inference made from circumstantial evidence. They recommend that courts should not follow literary cases by assuming that no direct evidence of copying is available and disposing of the case solely on the basis of substantial similarity. Davidson, for example, suggested that

341 807 F.2d 1256 (5th Cir. ); the court here found that similarities in sequence and organisation were dictated by the cotton market for which they were designed.


with proper expert testimony, it should be possible to argue successfully that source code changed in symbolic representation to disguise copying contains the same expression of the original. An example was *Midway Mfg. Co. v Strohon*\(^{345}\) in which substantial similarity was grounded in a finding of 89% identity at the machine code level. Another was *Williams Elecs. Inc. v Artie Int’l Inc.*\(^{346}\) where in addition to significant object code identity there was evidence of identical errors in the program as well as the plaintiff’s copyright notice buried in the object code. In the landmark case of *Apple Computer Inc. v Franklin Computer Corp.*\(^{347}\) the name of the Apple programmer was found embedded in the code of the defendant’s programs.

The major issue for interpretation of the copyright legislation is in making a clear distinction between the similarity in function or the “total concept and feel” of two programs which derives from defendant independently writing a competing program and that which derives from access to the plaintiff’s source code or a detailed description of that code. It is a contention of this thesis that source code and object code alone should be protected under copyright. Non-literal aspects or abstractions of function or interface, if protected at all, should be by patent or by copyright

---

346 685 F.2d 870 (3d. Cir. 1982).
347 714 F.2d 1240 (3rd Cir. 1983).
subsisting in the artistic or literary content embedded in the program, such as the cinematograph films implicit in computer games, or the text of a literary work distributed in CD-ROM. Some indications of courts moving in this direction are discussed in section 7.6, below.

7.5 Fair use exception

This section examines the extent to which fair use provisions of copyright legislation authorise reverse engineering activities with respect to software generally and, in particular, to decompilation. Fair use is an exception to exclusive rights and a defence to infringement.

Graham and Zerbe argued that reverse engineering of computer software should be allowed, and provide four main economic justifications. First, a prohibition against reverse engineering allows the innovator to prevent others from accessing the unprotected ideas underlying the creation of the work. Secondly, reverse engineering of ideas, but not expression, does not conflict with the mandate of the copyright legislation, eliminating the "excess" protection software producers claim they deserve. Thirdly, reverse engineering as a policy question is justified by the functional nature of software and the close analogies to patent. Finally, permitting reverse engineering provides a means of access where no other means are available.

Part of the learning process in reverse engineering can be disassembly (decompilation), which may be of some use in understanding small elements of the overall program, such an interoperability with the systems software or telecommunications functions. Kreiss\textsuperscript{349} argued that the doctrine of fair use should permit disassembly for study purposes. Similar calls have been made for a right for users to modify lawfully acquired software.\textsuperscript{350} There has been a typical assumption that access necessarily involves disassembly, which is itself assumed to be an infringement.\textsuperscript{351} In the light of recent United States decisions, some commentators have argued that where disassembly is a necessary step in ensuring compatibility, such intermediate copying should not be infringement.\textsuperscript{352} This proposition is in agreement with the policy developed in Chapter Six.


7.5.1 United States

In the United States, fair use is in section 107 of the Copyright Act, which provides that fair use means use of a copy for purposes such as "criticism, comment, news reporting, teaching, scholarship, or research". The United States courts have ruled that fair use authorises the making of interim copies and also decompilation for purposes of reverse engineering where this is essential for interoperability. _Atari Games Corp. v Nintendo of America Inc._ Federal Circuit ("Atari") explicitly held that the making of interim copies for study as part of the reverse engineering process was not infringement but fell within the fair use exception.\(^354\) The court in _Atari_ also stated: \(^355\)

> An author cannot acquire patent-like protection by putting an idea, process, or method of operation in an unintelligible format and asserting copyright infringement against those who try to understand that idea, process, or method of operation.

In _Sega Enterprises Ltd v Accolade Inc._\(^356\) ("Sega") Accolade manufactured video game cartridges and had considered becoming a licensee of Sega. This licence was not pursued since Sega's conditions would have required that Sega be the exclusive manufacturer of all Accolade games. Instead Accolade reverse engineered the Genesis game console interface

---

353 975 F.2d 832 (Fed. Cir. 1992)
354 *Ibidem* at p. 843.
355 975 F.2d 832 at p. 842.
356 977 F.2d 1510 (9th Cir. 1992)
specifications and manufactured Sega compatible video games. The Ninth Circuit ruled that Accolade was entitled to disassemble parts of Sega’s code in order to determine the necessary compatibility requirements of the Sega Genesis, and reproduce a short segment from Sega’s compatibility code, known as a “TMSS initialization code” in their own cartridges. In evaluating the question of fairness the court emphasised commercial use of the reverse engineering was indirect and the public benefited from dissemination of additional creative works. Both Sega and Atari support the principle that an author cannot expect copyright protection, which has the goal of dissemination of ideas, while at the same time attempting to keep the public from extracting these ideas.357

Similarly, in Lewis Galoob Toys Inc. v Nintendo of America Inc.358 the court upheld fair use by the defendant in developing a hardware device for improving a cartridge game machine. The “game genie” altered the behaviour of the Nintendo games, in particular allowing players to customise their difficulty level and a range of “cheats” such as proving additional “lives” to prolong a game. In part the court believed the use was fair given evidence that the device did not have an adverse economic

358 964 F.2d 965 (9th Cir. 1992).
impact on Nintendo, but instead enhanced sales.359

7.5.2 Europe

In contrast to the United States and Australia, where reverse engineering policy has been articulated in the courts, the EC has adopted a "top-down" policy which promotes the explicit recognition of the reverse engineering process within the EC Directive.

Article 5 explicitly permits a legitimate user to "observe, study or test the functioning of the program" in order to determine its ideas or principles. This endorses a model of innovation consistent with that set out in this thesis, that is, imitative competition by observation of rival products without access to the underlying source code. This is also described as "black box" reverse engineering.

Article 6 of the EC Directive creates an explicit reverse engineering exception permitting decompilation, reproduction and adaptation where these actions are:

... indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs.

The express intent of the exception is to encourage the development of open systems, to make it possible:360

359 Ibidem pp. 968-969.
360 EC Directive, p. 43.
... to connect all components of a computer system, including those of different manufacturers, so that they can work together.

Under Article 6 a reverse engineer using decompilation will have to demonstrate that this was “indispensable” to achieving interoperability. Implicitly, this demonstration would include an explanation of why observation of the operation of program was insufficient. Authorisation codes such as those in the Sega and Atari cases are likely to fall within Article 6, as are specifications for internal interfaces such as software-software and software-hardware protocols. Bainbridge\(^{361}\) stated that Sega is consistent with the EC Directive and with pre-existing United Kingdom laws. User interfaces, however, are more likely to fall within Article 5. Karjala\(^{362}\) argued that the EC Directive represented a political compromise which was flawed particularly in limiting reverse engineering practices to purposes of interoperability, for example in disassembly. It is arguable that this interoperability includes interoperability with the systems software as well as with peripherals such as printers and storage devices, but possibly not user interfaces.

Also important is Article 9 which required that contractual provisions which were contrary to the reverse engineering practices set out in


Articles 5 and 6 were to be rendered null and void. This was a particularly important element of the EC Directive to ensure that copyright holders did not exploit licensing agreements to overcome these exceptions.

One effect of the EC Directive has been the more explicit recognition of the rights of competitors in removing protection from interfaces and authorising access to internal interfaces by means of disassembly. In part this was driven by a different legal civil law tradition within Continental nations from Anglo-American law, preventing application of the fair use doctrine. Instead, the EC Directive was stated in explicit terms, albeit based on analogy to fair use approach.

7.5.3 Australia

Against the precedent in both the United States and Europe, the Australian High Court decision in *Autodesk Inc. v Dyason*\(^{363}\) extended the scope of copyright infringement effectively to prohibit the form of reverse engineering used in that case. In doing so it overturned the decision of the Full Federal Court which held there to be no infringement.\(^{364}\) The suit alleged infringement of a computer aided design program, AutoCAD. As a means to prevent piracy, the program required the presence of a hardware lock, AutoLock, a circuit which acted as a shift register which responded to challenges from the computer

---


\(^{364}\) *Dyason v Autodesk Inc.* (1990) 24 FCR 147; overturning the first instance ruling of
program every few seconds while in operation by returning an appropriate response. The register was based on a 127 bit string and associated routine contained within the code of the AutoCAD program, described by the court as Widget C. If the lock returned the correct response the program continued to function. Kelly, a computer expert, did not open the hardware device or examine the code of the program but instead used an oscilloscope to observe the challenges and responses to AutoLock and created a “look-up table” which mimicked the behaviour of AutoLock. Having done this he manufactured AutoKey which he then distributed to users of AutoCAD.

The High Court held that Widget C, as a subset of the AutoCAD program, in itself constituted a protectable computer program, and that copyright was infringed by the manufacture of AutoKey since AutoKey contained a copy of the 127 bit look-up table. The court found that this string was a “substantial part” of Widget C in the sense that it was essential for the operation of AutoLock, hence qualitatively substantial despite being quantitatively insignificant. In finding the look-up table protectable the High Court took the decision in Whelan as persuasive. Applying Whelan, the “idea”, of function, of Widget C was the protection of the AutoCAD program. The “expression”, or method for arriving at that function, was the use of a locking device employing a look-up table. The particular form of the look-up table chosen by Autodesk therefore

constituted protectable expression.

If this approach was applied by the High Court to the facts in the Sega case it would find infringement. The TMSS initialisation code, although quantitatively small would be ruled as qualitatively substantial since in that case the code was essential for operation of the game console, just as the 127-bit look-up table in Autodesk, was essential to the operation of Widget C. The approach in Autodesk is therefore inconsistent with the policy objectives developed in Chapter Six.

In Australia, the Copyright Law Review Committee (CLRC) in 1993 endorsed the approached adopted in the EC and Sega.\(^\text{365}\) In its 1995 report on Computer Software Protection it recommended the extension of the fair dealing provisions to include explicitly decompilation where necessary to achieve interoperability with software or hardware, based on the text of the EC Directive.\(^\text{366}\) Unfortunately the CLRC accompanied this measure with a proposal to render other purposes for decompilation, except for error correction\(^\text{367}\), *per se* illegal. These would include:

- enhancing program performance;\(^\text{368}\)

---


\(^\text{367}\) *Ibidem* para. 2.27, although this itself is limited to where an error free version of the program cannot be obtained within a reasonable time at a normal commercial price.

\(^\text{368}\) *Id.* para. 2.24.
networking a computer program;\textsuperscript{369}

porting a program to another platform;\textsuperscript{370}

circumventing a program lock;\textsuperscript{371} and

understanding the program’s underlying techniques.\textsuperscript{372}

This final prohibition has been the subject of particular criticism\textsuperscript{373}, and is inconsistent with the EC Directive. In addition, reverse engineering not involving decompilation – “black box” reverse engineering – should be limited, according to the CLRC, to “non-commercial” activities.\textsuperscript{374}

Based on the High Court decision, and the recommendations of the CLRC, the use of fair dealing exceptions as a basis for reverse engineering will continue to be severely limited in Australia.

7.6 Current case law

This section examines current case to assess the approach taken with respect of subsistence of copyright in “non-literal” elements of computer programs. These cases use different methodologies for applying the idea-

\textsuperscript{369} Id. para. 2.25.

\textsuperscript{370} Id. para. 2.26.

\textsuperscript{371} Id. para. 2.29.

\textsuperscript{372} Id. para. 2.28.


\textsuperscript{374} Ibidem para. 2.28.
expression distinction and recent litigation in the United States and Australia raise similar issues which remain unresolved in both jurisdictions. European law has yet to encounter serious litigation interpreting the EC Directive although the indications from recent cases in the United Kingdom\textsuperscript{375} suggest that the lead of the United States courts may be critical for future developments.

7.6.1 United States

In \textit{Lotus Dev. Corp. v Paperback Software Int’l}\textsuperscript{376} ("Paperback") the defendants admitted that they had intentionally imitated the user interface of "Lotus 1-2-3" in their competing program "VP-Planner". At the centre of the interface was a tree of menu commands from which the user could select and which included prompts explaining each command. It was accepted, however, that the VP-Planner program consisted entirely of new and original source code. Paperback argued that the user interface of 1-2-3, which at that time was the industry standard, was an uncopyrightable idea which could be legitimately used to develop a competing program. The court found that the logical organisation of the user interface of 1-2-3 was copyrighted. The key element of this logical organisation was the use of the slash (/) key to activate spreadsheet commands located in a series of menus. The user first activated a menu

\textsuperscript{375} Section 7.6.2.

Chapter Seven – Evaluation of International Copyright Laws

and then chose a command from within this menu. These commands were organised into a logical “tree” over which the user moved a cursor to select the appropriate command.

Since there were a number of ways in which to express the idea of a menu of commands to the user, competitors should be prohibited, according to the court, from imitating the particular menu structure adopted in 1-2-3. The court found, however, that the expression of the idea of an electronic spreadsheet as a rotated “L” with letters and numbers designating columns and rows was limited and so unprotectable. Also, given the limited range of keys on a computer keyboard, the use of the slash (/) key to invoke the menu system, or the “+”, “-”, “*” or “/” characters to represent arithmetical functions was not protected.

It might be noted that in making its findings the court disregarded expert evidence and legal articles on the issue, as well as rejecting the evidence of Daniel Bricklin, the inventor of the electronic spreadsheet who had argued against copyright for non-literal elements of programs. Despite the explicit statements in the judgment to the contrary Keeton J appeared to apply the Whelan approach in reaching its conclusions. Some statements in the judgment are difficult to interpret in this context.377 The wake of Paperback appeared to leave the state of the law in the United

States on non-literal protection in confusion.378

The same judge (Keeton J) ruled at first instance in *Lotus Development Corp. v Borland International Inc.*379 ("Borland") that a copy of the Lotus menu tree infringed Lotus' copyright even when it was invisible to the user and not part of the visual user interface. At issue was the similarity between 1-2-3 and Borland's Quattro and Quattro Pro user interfaces, including the menu structure and organisation, long prompts, function key assignments and the macro380 commands and associated macro language. The approach was again implicitly that of Whelan. Having determined the "idea" of the interface - a structure for a hierarchical system of menus which facilitated access to the spreadsheet operations - the specific menu structure of 1-2-3 was found to be protectable expression. This was based on the finding that the names of the commands and menus themselves (such as "Worksheet", "Range", "Copy", "Move", "File" and "Print") could have been renamed and reordered in "at least hundred and perhaps thousands" of ways.


380 "By writing a macro, a user can designate a series of command choices with a single macro keystroke. Then, to execute that series of commands in multiple parts of the spreadsheet, rather than typing the whole series in each time, the user only needs to type the single pre-programmed macro keystroke, causing the computer to recall and perform the designated series of commands automatically." See *Lotus Development Corporation v Borland International Inc.* 49 F. 3d 807 (1st Cir. 1995).
generating some “millions of satisfactory menu trees.”\textsuperscript{381} That the ordering in the menu was based on expected frequency of use of the various commands was not seen as a significant functional limitation on the arrangements of commands and menus.

Keeton J rejected Borland’s argument that the menu structure was a program element dictated by software compatibility, on the basis that the menu structure was linked to the 1-2-3 macro language. Rather, it was held that since the menu structure preceded the macro language, and since this structure was arbitrary, not dictated by efficiency, the structure was copyrightable, even if all the user macros derived from it.\textsuperscript{382}

In close analysis of the cases, commentators such as Carleton\textsuperscript{383} have argued that the test articulated by Keeton J in Borland is not substantively compatible with the Altai approach. In particular the Keeton J test focused on the issue of idea-expression merger and ignored the application of the \textit{scènes à faire} doctrine, or the examination of external factors such as interoperability or industry practice.\textsuperscript{384} The result of the application of the Keeton J approach is likely to undermine the objectives of compatibility and standardisation in user interfaces. Which element of

\textsuperscript{381} ibidem pp. 217-218.

\textsuperscript{382} 799 F. Supp. at pp. 213-214.


\textsuperscript{384} Brief Amicus Curiae of Copyright Law Professors in \textit{Lotus Dev. Corp. v Borland Int’l}
1-2-3 was created first should not be relevant. According to Carleton:\textsuperscript{385}

What matters in this case is that [Lotus'] macro system was created prior to the Borland interface, and thus became a pre-existing condition which limited the manner in which Borland could implement the interface which was to read and execute those macros.

The clear implication of both Keeton J rulings in the Lotus litigation was that copyright can be used to prevent a competitor from producing compatible software. Interfaces, even if \textit{de facto} standards, were rendered proprietary by these cases.\textsuperscript{386} As a question of policy Keeton J accepted the Lotus propositions in these respective cases that Paperback and Borland were competitors free riding on Lotus' innovations and efforts in making 1-2-3 the industry standard.\textsuperscript{387}

On appeal to the First Circuit,\textsuperscript{388} however, the court ruled in favour of Borland, and held that the menu command hierarchy was not protected by copyright. Importantly, the court found that \textit{Altai} did not apply to this particular situation. The situation was not one of "non-literal" copying but of "literal" copying of the menu command hierarchy. The issue for


\textsuperscript{388} \textit{Lotus Development Corporation v Borland International Inc.} 49 F. 3d 807 (1st Cir. 1995).
the court was not the application of the "abstraction-filtration-comparison" test, but whether a menu command hierarchy is itself protectable. The court suggested that application of Altai would "obscure" this more fundamental question. Indeed, the court was critical of using traditional copyright approaches to computer programs "in cookie cutter fashion, as if the programs were novels or play scripts", and felt that courts which had done so were mistaken.

The First Circuit agreed with Borland's argument that the menu command hierarchy is uncopyrightable as a "method of operation" excluded from protection by s. 102(b) of the United States legislation. This was based on the finding that the command hierarchy provides the means by which users control and operate Lotus 1-2-3. An analogy was drawn between the 1-2-3 menu command hierarchy and the buttons used to control a video cassette recorder (VCR):

That the buttons are arranged and labelled does not make them a "literary work", nor does it make them an "expression" of the abstract "method of operating" a VCR via a set of labelled buttons. Instead, the buttons themselves are the "method of operating" the VCR... Without the menu commands, there would be no way to "push" the Lotus buttons.

In its decision the court clearly linked the issues of program compatibility and standardisation to the question of whether a menu hierarchy is protectable. The court rejected as "absurd" the implication from Lotus' arguments that a user should learn how to perform the same operation

389 The court appeared to disagree with Gates Rubber Co. v Bando Chem. Indus. Ltd 9 F.
(such as printing material) in a different way for each different program they were using. It was also seen as inappropriate to force users who had written macros for 1-2-3 to rewrite these macros to suit a different command hierarchy. There was also explicit recognition of the danger to incremental innovation if the Lotus submission was upheld:

"[B]uilding" requires the use of the precise method of operation already employed ... Original developers are not the only people entitled to build on the methods of operation they create; anyone can. Thus, Borland may build on the methods of operation that Lotus designed and may use the Lotus menu command hierarchy to do so.

Keeton J's application of the idea-expression distinction was criticised in drawing the line between idea and expression at too high a level of abstraction and thereby providing too broad a scope of protection:

We do not think that "methods of operation" are limited to abstractions; rather, they are the means by which a user operates something. If specific words are essential to operating something, then they are part of a "method of operation" and, as such, are unprotectable.

As a matter of policy, competitors should not be foreclosed from using the specific command terms and hierarchical arrangement that Lotus had used. In drawing an analogy to the QWERTY keyboard, users who had learned the command structure of 1-2-3 and written their own macros would be locked into Lotus if Borland were prohibited from using the same pattern. This would provide undue market power to Lotus and fail to reward Borland for the value of its own innovations.

3d 823 (10th Cir. 1993).
Borland is important for implicitly drawing a distinction, as argued above, between situations of misappropriation of another program's code and imitation of another program's features using freshly written code. This thesis argues for an interpretation of the Borland decision such that the Altai test applies to the former while the Borland analysis (echoing Synercom) applies to the latter. As the First Circuit stated:

While the Altai test may provide a useful framework for assessing the alleged nonliteral copying of computer code, we find it to be of little help in assessing whether the literal copying of a menu command hierarchy constitutes copyright infringement.

The implication would be that, if there has been no access to the source code of the plaintiff, the issue for the court is whether a particular feature of the program is, of itself, a protectable work for the purposes of copyright. The court effectively endorsed the approach that, provided a competitor write their own code, there should be little prohibition on imitation of function or interfaces:

[T]o offer the same capabilities as Lotus 1-2-3, Borland did not have to copy Lotus' underlying code (and indeed it did not); to allow users to operate its programs in substantially the same way, however, Borland had to copy the Lotus menu command hierarchy.

The appeal decision in Borland agrees with the policy objectives developed in this thesis, and also agrees with the distinction between approaches based on copying of code and imitation of features with fresh code. There are, however, some reasons to be concerned that the

390 Emphasis added.
precedent set by *Borland* might not be a lasting or applicable to other countries. One issue is that the First Circuit upheld the argument as to “method of operation” but did not consider the question of whether it was also a “system, process or procedure”. It is not clear that the court’s decision will be adopted by courts outside the United States using only general law principles of the idea-expression distinction. Some indication of a willingness to do so is found in the Australian decision in *Powerflex*.391

The decision of the First Circuit was also at odds with an earlier decision of the Tenth Circuit in *Autoskill Inc. v. National Educ. Support Sys. Inc.*392 which rejected a similar argument based on s. 102(b).

More significantly, the appeal to the Supreme Court in *Borland* resulted in an equally divided (4-4) court, announced on 16th January 1996, with no written judgments supplied. The major issues in the appeal squarely addressed the concern of this thesis. For example, the opening sentence to Lotus’ argument in its petitioner’s brief to the Supreme Court stated:

> The issue in this case is whether the copyright protection Congress granted computer programs protects the original, creative expression contained in their user interfaces against wholesale appropriation.

That this proposition obtained support from four of the nine serving judges suggests that it is possible that in a further appeal, user interfaces

391 Section 7.6.3.
392 994 F.2d 1476 (10th Cir. 1993).
such as the Lotus 1-2-3 menu command hierarchy may become per se protectable. This would be likely to flow onto other jurisdictions such as Europe which, as discussed below, have indicated some willingness to following the lead of the United States courts.

7.6.2 Europe

In Europe, it should be noted that the EC Directive restated the idea expression dichotomy in that the principles underlying a computer program are not protected. Moreover the EC Directive also states that the principles underlying a program’s interfaces are also not protected.

Courts in EC member states will, however, be faced with the task of developing jurisprudence to deal with these copyright law changes over the coming years. This may result in some “lurching, backtracking and zigzagging”\(^{393}\) before a general consensus is reached. Cases in the United Kingdom, both before and after the EC Directive have, however, indicated a willingness to be led by the United States courts in developing this jurisprudence.

In *Computer-Aided Systems (UK) Ltd v Bolwell*\(^{394}\) the plaintiff’s ex-employees wrote a fourth generation language program, Progress, which had similar functions to a vehicle leasing program in COBOL written for

---


\(^{394}\) (Unreported, Chancery Division, 23 August 1989).
the plaintiff. Hoffmann J cited *Whelan* with approval, and held that program structure was a form of literary expression protected by copyright. The significantly different conceptual characteristics of the fourth generation language, however, did not suggest an objective similarity between the programs.

*John Richardson Computers Ltd v Flanders*[^395] dealt with programs for labelling medicines. The defendant had been employed by the plaintiff to modify and improve the plaintiff’s program written in BASIC, later rendered in machine code for the BBC microcomputer by rewriting in assembly language. Subsequently, the defendant developed a rival program which performed similar functions with more advanced features, although it was written in QuickBASIC for the IBM. Given no similarity in source code (between BBC assembler and IBM QuickBASIC) the court assessed similarity in non-literal elements. Of 17 similarities 3 (the line editor, amendment routines and dose codes) were found to have been the result of copying the plaintiff’s program and infringement upheld. Importantly, Ferris J cited *Altai* with approval, stating:

> There is thus nothing in any English decision which conflicts with the general approach adopted in the Computer Associates case . . . it would be right to adopt a similar approach in England. This means that consideration [of copying] is not restricted to the text of the code.

In Ibos Computers Ltd v Barclays Mercantile Highland Finance Ltd\(^{396}\) a program called ADS had been jointly developed between programmers P and C over a number of years after which P separately developed a competing program called Unicorn. An inference of copying was drawn from the presence of common spelling mistakes, identical comment headings, file records and redundant and unexplained code in the source code of both programs. The court found infringement on the basis that the defendant had taken short cuts in creating his program by starting with ADS and making modifications to produce Unicorn. It should also be noted that Jacob J raised caution about the direct applicability of United States cases to English law.

### 7.6.3 Australia

The situation in Australia will remain uncertain until the High Court rules on the pending appeal from the Full Federal Court in Powerflex Services Pty Ltd v Data Access Corporation\(^{397}\). The Powerflex litigation paralleled the Borland litigation in the United States. The approach and findings of the court at first instance\(^{398}\) on appeal in both cases was remarkably similar.

Data Access owned copyright in the program DataFlex, which provided a


\(^{397}\) [1997] FCA 490.

set of programming tools to develop database management systems. The program was comprised of a development component (compiler tools) and a runtime system (interpreter). The Dr Bennett was a user of DataFlex for a number of years and desired to create an application development system which would be compatible with the DataFlex program in terms of using the same commands, file structure and functions keys. By study of the manuals and close observation of the operation of DataFlex, Dr Bennett independently wrote his own program which was initially called Powerflex although later changed to PFXplus, which was sold commercially as a competitor to DataFlex. There was no similarity in source code or object code of the two programs.

Infringement was alleged on the basis that PFXplus "language" (the set of command words and macros used) was copied from the DataFlex program, along with a compression algorithm look-up table, identical function key assignments, and an error text table. The key issue was whether the command words individually or together constituted a computer program within the meaning of the Act. If the command words were protectable, then infringement would have followed as some 192 of the 254 Dataflex words were contained in PFXplus.

Jenkinson J at first instance held that each one of the words in the DataFlex language was protected by copyright as a computer program. Each word constituted a "set of instructions" which caused the computer to perform a particular function. The declarative words and phrases of
the PFXplus program (expressions in a fourth generation language) were converted to a set of procedural instructions in C (a third generation language) then to machine code for execution by the runtime system. That there was no similarity in the source code underlying the commands words was seen as immaterial.

Powerflex argued that the words constituted unprotectable “ideas” within accepted copyright doctrine and drew upon the decision in *Borland* and in particular its statement that:\(^{399}\)

> If specific words are essential to operating something, then they are part of a “method of operation” and, as such, are unprotectable.

The principle in *Borland* was that words such as “Copy” and “Print”, as used in the menu hierarchies, were uncopyrightable in contrast to the underlying code which was protected expression. Jenkinson J rejected these arguments on two bases. The first was that he distinguished *Borland* as an application of s. 102(b) of the United States’ legislation and the phrase “method of operation” which did not exist in the Australian Act. The second was the application of Keeton J’s approach to infringement in *Paperback*, which, as discussed above, was based on the *Whelan* principles. He held that the words in the DataFlex language were protected expression on the basis that there were numerous other ways of expressing the functions which the commands performed. Even though the words were mnemonic and suggested the function to be performed,

---

\(^{399}\) 49 F.3d. 807 (1st Cir. 1995) at p. 816.
there were other words which could have been used to execute these functions. On the same basis, each of the allocation of 16 words to function keys was a computer program and their imitation by the defendant an act of infringement.

In examining the macros the court did find similarity in the source code underlying the macros, although the methodology employed by the court was questionable. The DataFlex macros were written in an intermediate code (analogous to assembler language) while the PFXplus macros were written in source code. The court made a comparison between the PFXplus source code and a rendering of the DataFlex intermediate code into source code which the court accepted as a “translation”. As discussed in Chapter Five, there would have been little isomorphic relationship between source and intermediate code and the reliability of this subjective exercise from an evidentiary perspective is speculative. Fong argued that the only real similarity between the two sets of macros was in their function.

Powerflex did not infringe, however, in the reproduction of an error text table, a look-up table of 100 strings of text responses when the program recognised an error in function. Jenkinson J accepted the argument that the number of ways in which the error table could be expressed was limited and that the expression merged with idea. Surprisingly, the court

---

also held that reproduction of a Huffman compression table did infringe. The table was based on a compression algorithm whereby characters normally encoded in 8 bits were encoded by using short strings of less than 8 bits for common characters, and for uncommon characters using longer strings. The result is a significant saving of storage space. The court appeared to base its ruling on the acceptance of evidence that the table might have been constructed in some alternative ways using the same compression algorithm. On the same basis, the use of the DataFlex file structure infringed, even though this was essential if files created in one system were to be compatible with the other. The PFXplus file structure instructions were treated as an adaptation of the Dataflex instructions, even though there was no similarity in code.

Effectively, the decision at first instance granted an Australia-wide monopoly to Dataflex over use of 192 command words in computer programs, including such common words as “Print”, “Display”, “Save”, “Find” and “Help”, common words which existed in other languages developed before DataFlex, such as BASIC. The first instance decision was severely criticised as providing greater protection to American companies in Australian than that which they would receive in the United States, putting Australian companies at a distinct competitive disadvantage.401

On appeal, the Full Court overturned the decision and found that the DataFlex words were not computer programs:

Each of the words in the so-called DataFlex language is but a cipher. The underlying program is the set of instructions which directs the computer what to do when that cipher is in fact used, for example by being typed on to the screen. It is not to the point that the cipher bears some resemblance to an ordinary English word. The cipher or command is not an expression of the set of instructions, although it appears in that set of instructions. It is the trigger for the set of instructions to be given effect to by the computer.

The Full Court identified the computer program as being expressed in the source code, not in the choice of command words. Unlike Jenkinson J, the Full Court endorsed the approach of Borland and noted that the Court of Appeals in Borland stated that s. 102(b) of the United States' Act added little to the general law principle of separation of ideas and expression. The Supreme Court judgment in Baker v Selden402 was seen as embodying this principle in common to both jurisdictions. Given the description of the command words as “ciphers”, the court refused to protect the words themselves since they were ideas rather than expression. Extending this argument, both of the function key words and the macros were also unprotectable:

In our view, a command which triggers the underlying computer program to perform more than one function is no more a computer program than is a command which triggers the program to perform only one function.

The court rejected the objective similarity between the two versions of

source code underpinning the macros as justifying a finding of an "adaptation" of the DataFlex macros:

In our view, a process of devising a source code to perform the same function as is performed in some other source code expressed in original language does not involve creating a version of the original source code. It is an original work, albeit that the function to be performed is the same.

The court agreed that the error table was not infringed, although not on the grounds of merger held in first instance. The table might be protected as a table or compilation (not a computer program), but on the evidence there were sufficient differences in the texts to prevent a finding of reproduction. The court did uphold, however, the infringement by reproduction of the Huffman compression table, although again as a table or compilation and not as a computer program. While Dr Bennett did not directly copy the compression table, by observation of the behaviour of the DataFlex program and use of test files the PFXplus compression table was effectively reverse engineered, although its final form was indistinguishable from that of DataFlex. On this basis, echoing the earlier High Court ruling in Autodesk v Dyason, the table had been indirectly copied by this reverse engineering process and therefore infringed.

The court also overturned the finding that the use of the file structure infringed. With no similarity in source code between the two programs the only identity was that of function, which was unprotectable. Again the court rejected the description of PFXplus as an "adaptation" of DataFlex in this respect.
The approach of the Full Federal Court, like that of the First Circuit in *Borland*, agrees with the policy prescriptions in this thesis. In particular, both courts appear to recognise that copyright principally protects program code and that protection of "non-literal" elements should not be equated with protection of the underlying code. Both have emphasised the importance of interfaces and refused to protect elements of these interfaces based on analogous reasoning. In *Borland*, the menu command hierarchy was an unprotectable "method of operation". In *Powerflex*, the database language was an unprotectable "cipher". The major difference is that Australia is likely to have a ruling shortly from its highest appellate court which will settle the law, whereas in the United States the split decision of the Supreme Court is likely to generate further uncertainty.

### 7.7 Political considerations

In assessing the current state of the law internationally it is important to remember that legislation and case law did not arise in a political vacuum. While some of the policy prescriptions developed within this thesis may represent a normative policy objective, desirable innovation policy in the information economy needs to be tempered by an understanding of the political economy. Legislative reform may be impeded or circumvented by powerful interest groups actively participating within the policy development and legislative process.
Indeed, Litman\textsuperscript{403} has argued that the nature of the legislative process in the United States was to blame for difficulties in accommodating copyright law for new technologies. Olson\textsuperscript{404} believed that Congress as a political body was loathe to implement changes which would result in one-sided losses on interest groups, with the result that “good copyright policy” was often delayed or abandoned. Menell\textsuperscript{405} noted that one of the difficulties was that policymakers did not have a sufficient understanding of the path of information technology and the implications for a desirable intellectual property regime. He suggested that the outcome of the CONTU process was unsurprisingly, “a rather naive set of recommendations, closely approximating the status quo”.\textsuperscript{406}

One example of politics affecting intellectual property policy was the development of the EC Directive. Political considerations provide one explanation for the significant concessions contained in the EC Directive to reverse engineering and access to interfaces. From December 1989 to July 1990, the European Parliament considered the issue of reverse

\begin{flushright}
\textsuperscript{406} Ibidem p. 2652.
\end{flushright}
engineering within the EC Directive in a debate which was:407

... often heated, sometimes illuminating, never conclusively persuasive one way or the other.

A "virtual firestorm erupted"408 dividing the European computer industry into two opposing camps. The European Committee for Interoperable Systems (ECIS) was formed in September 1989 and included companies such as Bull, Olivetti, Fujitsu, NCR, UNISYS and Sun. ECIS supported an open interface approach and endorsed the freedom to conduct reverse engineering and extraction of interface specifications. Against ECIS rose the Software Action Group for Europe (SAGE) which included IBM, DEC, Apple, Lotus and Microsoft.409 SAGE opposed the draft directive arguing for broader protection of software to fight "piracy" and to stimulate investment. About a year later, the Computer Users of Europe (CUE) emerged to argue for users' concerns in support of freedom of competition and interoperability. Vinje410 noted that SAGE and ECIS argued from the perspective of their membership's situation with respect to de facto interfaces. IBM's Systems Network Architecture and


Microsoft's MS-DOS, for example, established dominant standards to which the members of ECIS felt "locked-in", as access to and use of which were tightly controlled by members of SAGE. At one level the situation in Europe reflected concerns about fostering local technological industries to compete internationally, particularly a growing European IBM-compatible PC industry led by Bull and Olivetti. The key issue was the ability to create products compatible with the PC-BIOS. ECIS insisted that acceptance of counter-proposals made by SAGE would effectively insulate American hardware and software companies from European competition.\footnote{Ibidem p. 62.} Goldstein\footnote{Goldstein, P. (1993) "The EC Software Directive: A View from the United States of America" in Lehmann, M. and Tapper, C.F. (eds) A Handbook of European Software Law (Oxford: Clarendon Press) p. 203 at p. 205.} noted that a prohibition on seeking compatibility through legitimate research and decompilation would be:

\begin{quote}
... at the expense of Community-based companies seeking to make their products compatible with products coming from the United States. Not only would Community consumers pay the difference; a large part of the price paid would flow out of the Community.
\end{quote}

A major policy objective of the EC was the harmonisation of telecommunications and the removal of barriers to harmonisation such as the existence of technical requirements of local communications networks inconsistent with international standards. The identification of computer software as the key to the development of effective communications systems has in part led to similar policy needs for
regulation in these sectors. The explicit removal of protection to interfaces in the EC Directive echoed the requirement in the Directive on Liberalization of Terminal Equipment to disclose interface specifications. It is arguable that the EC’s long interest in the area of telecommunications interfaces and its experience with IBM’s refusal to provide European companies with access to interface specifications in the 1980’s naturally applied to the debate on software protocols. It is consistent with the above to note that the EC itself is a supra-national body principally concerned, under Article 2 of the Treaty of Rome, to promote development of the Community economy and has been a major contributor to the development of competition policy in Europe. Competition law serves as a policy instrument to shape and consolidate a Community-wide integrated internal market.

Australia, by contrast, represented a small country whose development of copyright legislation was driven principally by the concerns of multinational software houses and the United States government in applying unilateral trade pressures. For example, the CLRC had been


414 Directive 88/301EC Articles 4 and 5.


given a reference to investigate the legal protection of computer programs. Fearing potential change from the status quo the United States acted to pre-empt the Australian government on the issue. On 30 April 1993, shortly before the draft report on the CLRC was to be finalised, the United States Trade Representative Mickey Kantor signalled his intention to retain Australia on the “Priority Watch List”, under s. 301 of the United States Trade and Competitiveness Act 1974, on the basis that there was a risk that Australia might adopt a protection regime for computer programs other than as a literary works under copyright. It is arguable that the CLRC responded to this pressure. Its draft report recommended not only the continuation of protection for computer programs as literary works, but notably recommended replacing the Australian definition of “computer program” with the definition in the United States Act. On 30 April 1994, Australia was transferred from the “priority watch list” to the “watch list”. In its final report the CLRC stated:

The Committee would like to think that the Draft Report . . . served to reassure the US Government any remove any misapprehension that it might have previously had about the Committee’s intentions.

More important for the future of computer software copyright are

Australia” Law and Policy, Vol. 17, p. 283.

417 Together with the European Community, Argentina, Brazil, Egypt, Hungary, Korea, Poland, Saudi Arabia, Taiwan and Turkey.


419 Copyright Law Review Committee (1995) Computer Software Protection (AGPS:
growing pressures within international institutions to harmonise
copyright protection in a direction of strong protection favoured.
Principal amongst these are the *Berne Convention*, the Final Act of the
Uruguay Round and unilateral trade action by the United States. The
*Berne Convention* consists of a series of treaties dating from 1886 which
provides that signatory states to the Berne Union agree to accord an
author's works, originating in any member country of the Union, the
same protection it accords works originating in its own country. The
Convention also dictates some minimum requirements for protection.
The EC Directive was based on the *Berne Convention* and that its
provisions dealing with computer programs as literary works are
consistent is alleged specifically in Recitals 25 and 29 and also in Article
1(1). A significant exception to the permitted reverse engineering is article
6(2)(c) of the EC Directive which removes the fair use exception where the
purpose is the development of a computer program "substantially similar
in expression, or for any other act which infringes copyright". This may
potentially, depending on subsequent case law, include non-literal
infringement. The presence of Article 6(2)(c) was designed to ensure
that the EC Directive is consistent with the *Berne Convention*.421

---


421 Verstrynge, J-F. (1993) "Protecting Intellectual Property Rights within the New Pan-
The development of protection legislation for computer programs other than as literary works within the *Berne Convention* appears foreclosed internationally\(^{422}\) by the Trade Related Aspects of Intellectual Property Rights (TRIPS) as part of the Final Act of the Uruguay Round 1994. In particular, article 10(1) of the TRIPS agreement provides:

Computer programs, whether in source or object code, shall be protected as literary works under the *Berne Convention* (1971).

Further, Article 10(2) requires in part that:

Compilations of data or other material, whether in machine readable or other form, which by reason of the selection or arrangement of their content constitute intellectual creations shall be protected as such . . .

The difference between the *Berne Convention* and TRIPS, however, was that the GATT created a monitoring and enforcement mechanism through the establishment of the World Trade Organisation (WTO) to implement TRIPS. The WTO Treaty included dispute resolution procedures and authorised cross retaliation across different trade sectors. TRIPS was therefore a far more powerful instrument than the Convention in globalising intellectual property regimes in a direction favouring strong protection.\(^{423}\)


Ginsberg argued that following TRIPS there was no longer a realistic alternative to copyright protection for computer programs.424 Pressure was also placed on countries to confer specific additional rights such a rental right for literary works (including computer programs) required by Article 11 and 14(4) of the TRIPS Agreement. In Australia CLRC endorsed the TRIPS requirement by recommending the introduction of a rental right.425 Reichman and Samuelson426 have also demonstrated the influence of vested interests in using the TRIPS regime to strengthen copyright protection in recent legislative initiatives in Europe and the United States to provide strong monopolies over information compiled in databases.

7.8 Summary
This chapter has examined the current state of the law with respect to computer program copyright with particular focus on the United States, Europe and Australia. Of particular interest were the effectiveness of the idea-expression distinction and the doctrine of fair use in providing encouragement to the development of competing products given the importance of internal interfaces and user interfaces. Although Landes


and Posner had hoped that:\(^{427}\)

\[\ldots\text{ the debate will be resolved not by the semantics of the words "idea" and "expression" but by the economics of the problem.}\]

the courts have, as discussed in this chapter, generally focussed on drawing the line between protectable and unprotectable elements of computer programs through fixing idea, or expression, or both in a semantic fashion. Most clearly this was illustrated by the approach in *Whelan*.

In light of conflicting decisions it is difficult to come to a definite conclusion as to the scope of copyright protection for non-literal aspects of computer programs.\(^{428}\) This is particularly so given the 4-4 split by Supreme Court in the *Borland* case.

In terms of desirable copyright policy it is tempting to endorse an *Altai* approach and reject the *Whelan* approach. It is arguable, however, that *Altai* might better be described as a refinement of the *Whelan* analysis rather than a rejection of it.\(^{429}\) Both cases were dealing with alleged misappropriation of source code and related source materials. With this kept in mind *Whelan* used the similarity in structure, sequence and organisation of the two programs to provide evidentiary grounds for an

---


inference of misappropriation. Altai created a more sophisticated methodology for the same purpose. The major problem with the judgments was that both courts used the language of idea and expression to explain their approach. It is not surprising that in cases such as Paperback and Borland, where the situation was not misappropriation but of independent development, that the courts unconsciously elevated non-literal similarities from evidentiary facts to works protectable per se under copyright. It is suggested that resolution of the confusion surrounding non-literal copying will rely on courts making a clearer distinction between alleged copying of source code and alleged copying of function. An Altai approach would suit the former while the Synercom approach the latter.

The advantage of this recommended approach is that it provides a “bright-line” test for infringement in cases where there has been no access to the plaintiff’s source code and related materials. A bright-line test is needed if competitors are to be aware of what is and is not legal when developing a competing software product. A regime which forces all competitors to “play it safe” will have a negative effect on innovation, particularly in cross fertilisation of ideas, concepts and user interfaces.

Karjala argued that an idealised scheme of program copyrights would avoid legal standards requiring expensive and time-consuming litigation

---

Chapter Seven – Evaluation of International Copyright Laws

over difficult factual questions. It has been noted that the greatest attribute of the test in Altai is its flexibility and inherent discretion vested in the court in the filtering process.431 A major issue would be a factual examination of a program for comparison with efficiency consideration or a range of external factors including compatible hardware or software, and industry practices. All these issues may be the subject of dispute and require expert witnesses to resolve. This is clear in Lotus’ petitioner’s brief to the Supreme Court that the idea-expression compelled a “fact-based, line-drawing exercise” in every case of computer program copyright.

Simply put, an Altai approach would contribute to the uncertainty of litigation and place smaller companies at a disadvantage when dealing with multinational software houses.432 Recognition, however, that Altai is only appropriate in cases of misappropriation, and not where it is clear by other evidence that the defendant has had no access to the plaintiff’s source code, would avoid protracted and costly litigation. Cases such as Borland and Powerflex suggest that the legal issue will be limited to whether the non-literal element is itself a protectable work under copyright law.

Australia may be fortunate in that it is likely that the High Court will

---


resolve the issue for its domestic law. The United States will remain in a state of uncertainty until the Supreme Court has an opportunity to rule on what it believes Congressional intent to have been in respect of computer program copyright. In the short term the clear policy statements in *Borland* may discourage similar litigation. Europe has yet to articulate its interpretation of the EC Directive and will, like Australia, most likely follow the lead of the United States. Although the decisions in *Borland* and *Powerflex* appear to be moving in this direction, the author has some sympathy with the view that the system is “still badly bent, even if not completely broken”.433 In the background should not be forgotten the increasingly critical role international institutions such as TRIPS will play in the future development of intellectual property beyond computer programs, such as databases, multimedia technologies, and internet regulation.

---

Chapter Eight
Conclusions

8.1 Review

The major theme within this thesis was that information economics provided a distinct and preferable perspective to neo-classical economics in the analysis of technological development and therefore in the role of intellectual property in promoting innovation.

The neo-classical approach, grounded in Arrow's concerns about appropriability, indicated that legal intervention was necessary to correct a particular market failure: that innovators might be under-rewarded since the information they produced could be easily transmitted to those who were not compelled to pay for its use. Within this perspective, examined in Chapter Two, appropriability is achieved by the imposition of intellectual property rights. These rights generate an incentive to invest based on excluding access to information. This exclusion attempts to force information to mimic the commodity characteristics of tangible goods. From this assumption, it follows that the promotion of innovation is found in the maximisation of potential returns to innovators and consequentially the perfection of appropriability through broadening of intellectual property protection. Regulatory failure arising from monopoly distortions, underutilisation, wasteful research, and anti-competitive strategies is marginalised within this literature.
In contrast to the commodity view, an information economics perspective argued that information is preferably viewed as a resource. This conception of information, discussed in Chapter Three views it as part of shared technological capital, rather than as an industrial product. From this perspective, policy should seek to exploit the indivisibilities of information for social benefit in preference to maximising its appropriability. Moreover, innovation itself is not a bounded, independent process involving a single firm and capital investments, but a process which is evolutionary and interactive, involving many participants. Cumulative innovation through imitation and learning is dictated by bounded rationality in the face of limited knowledge and abilities of the economic actors. Limited information processing abilities and costs involved in learning new information means that the economy is likely to exhibit significant lead times such that investment in innovation can be rational without strong intellectual property protection. Informationally rich environments will also generate powerful network externalities, rewarding innovators with market power based on product leadership and bandwagon effects.

Based on this information perspective a model of innovation was developed in Chapter Four which markedly differed from the "reward for invention" model due to its contrasting underlying philosophy. Whereas the orthodox approach assumed that information displays "public good" characteristics – that others can access the information resulting in little reward for the innovator – the informational perspective assumed that
information does not exist as an abstraction but was reliant upon the limitations of human understanding and communication which can be embodied in two principal ways. Knowledge can exist in both a codified and a tacit form. The less codified the knowledge, the less it displays public good characteristics. The more tacit the knowledge, the more difficult that knowledge is to articulate and communicate. The degree to which one can appropriate another’s innovation is impeded both by the availability of the innovation in a codified form and the learning process involved in turning that codified knowledge into tacit knowledge. This is one explanation for the existence of natural lead times in innovation. Another way to embody information is to create technological products such as a physical machine or a piece of software. In the innovation model these products were defined as “information technology artefacts”. The usefulness of this concept is that it permits a distinction to be drawn between the economic characteristics of the underlying technological knowledge and the characteristics of the physical artefact. The latter, despite the innovation embodied therein, would be an excludable commodity and not the subject of market failure assumed in a neo-classical perspective. Assuming that a principal objective of intellectual property policy is the promotion of innovation, the IT artefact model of innovation can be used to explain the application the legal mechanisms of confidential information, patent and copyright.

Chapter Five demonstrated that computer programs are a particular form of technological innovation which reflects the characteristics of the IT
artefact. This was based in the proposition that access to the executable
form (machine code) of a program does not readily permit appropriation
of the underlying knowledge. Although this proposition stands at odds
with much of the legal literature in this area, it is justified by close
examination of the nature of the software development process which
shows, in particular, that when dealing with third or higher generation
language development, there is no isomorphic relationship between the
source code and the compiled machine code. Hence, decompilation is of
limited utility. The result of this is that the executable form of a computer
program corresponds to an IT artefact. The underlying source code
represents the codified knowledge articulated by the programmers from
their own tacit knowledge and by observation of other programs, in a
cumulative learning process. The close analogy between the software
development model and that of the confidential information scenario
suggested a limited role for intellectual property protection beyond
enforcement of trade secrets. A basic role for copyright is the prohibition
of literal copying or "piracy" due to the costless replicability of machine
code.

In critically revisiting arguments for broad intellectual property protection
for computer programs, the prescription of proprietary interfaces and
standards in some of the policy literature described in Chapter Six is
inappropriately based on a neo-classical perspective and inconsistent with
the realisation of powerful network externalities within the industry. An
information perspective instead recommended the encouragement of
reverse engineering and imitative competition on the basis that competitors implement their own source code solutions, thereby investing in the innovation process. Goals of efficiency and compatibility should not be hampered by intellectual property, so decompilation should be permitted to provide a limited degree of access to interfaces and communications protocols. Elements of a user interface should not be protectable per se.

The consistency of copyright regimes in the United States, Europe and Australia with the policy implication of the advocated information perspective was examined in Chapter Seven. The legal situation will continue to be uncertain until key rulings of the Supreme Court and the Australian High Court have been handed down. While this is shortly to take place in Australia, the tied decision in *Borland* means that it may be some years before the Supreme Court has a further opportunity to consider the issue. The law may remain unsettled in Europe for longer, given the issuing of the EC Directive. Courts in Europe may take some time to reach some consensus on the domestic interpretation of the EC Directive and are likely to be influenced by the United States. The key issue of dispute is whether “non-literal” elements of computer programs, such as “structure, sequence and organisation”, internal interfaces such as file structure and look-up tables, and user interfaces such as menu command hierarchies, are protected. The rulings of both *Borland* and *Powerflex* on appeal would, on the whole, negate this proposition. This negation is consistent with the objectives of an information perspective.
International political pressures such as TRIPS, however, may influence future developments in copyright legislation over information technologies in the direction of stronger protection.

8.2 Key Aspects and Recommendations

This thesis provides five key contributions to the understanding of copyright and intellectual property in computer programs. First is the use of information economics as a lens through which to view innovation policy and the application of intellectual property to information technology. In particular, this thesis extends the literature in information economics, and analogous analyses of patent issues,\textsuperscript{434} into copyright. The value of information economics lies principally in the insight it provides into the role of knowledge and learning within technological innovation. More importantly, it represents a significant counterpoint to the "reward" perspective dominant in neo-classical analysis. As Boyle\textsuperscript{435} stated:

\begin{quote}
[Incentive] analysis largely ignores the opposite perspective, that of the free flow of information. If we switch the perspective, we can see that one important purpose of [intellectual property] law is to make sure that future creators have available to them an adequate supply of raw materials. From this perspective, too many "incentives" could convert the public domain into a fallow landscape of private plots.
\end{quote}


Second is the development of a model of innovation which brings together two key information economics concepts regarding the embodiment of information. One is the recognition of the degrees to which knowledge exists in codified and tacit forms. This distinction was used by Mandeville\textsuperscript{436} in his analysis of the patent system. The other is recognition of the embodiment of technological knowledge within physical products, defined in this model as “IT artefacts”. This knowledge-artefact distinction was adapted from the work of Monk\textsuperscript{437} in his study of technological change. By combining these distinctions, the model of innovation represents information embodied in three ways: as codified knowledge, tacit knowledge, and IT artefacts. The model explains the role of intellectual property laws in promoting innovation through the flow of tacit knowledge.

Third is the rejection of a dominant assumption within the legal policy literature on computer program copyright – that possession of the software product (machine code) equates with access to the underlying knowledge (source code). This proposition is critical not only because the assumption of access has been the foundation for much of the legal literature favouring broader protection of programs, but it also facilitates


the application of the innovation model to software development.

Fourth is application of the innovation model to develop policy implications for intellectual property in the context of computer programs. It was argued that the law should reflect a model of confidential information, where imitative competition and reverse engineering promote technological innovation. The role of copyright law was in providing a basic level of protection to prevent literal replication of programs in their machine code form - "piracy". The proposition that copyright should provide broader protection over "non-literal" elements such as user interfaces was rejected.

Fifth is the reconciliation of the case law regarding computer program copyright with these policy objectives. The courts have failed explicitly to recognise that alleged copyright infringement has been in two forms. One is by misappropriation of source code, for which non-literal similarities provide circumstantial evidence of actual copying. The approach in Altai would be suited to this kind of complex factual investigation. The other is by competitors imitating successful features or interfaces by engaging in their own development process and writing fresh source code. The approach in Synercom, and the appeal decisions in Borland, and Powerflex would be more appropriate in providing a "bright-line" infringement test, whereby the court's task is a question, not of "non-literal" copying, but of whether the alleged feature or interface is itself protectable under copyright. It remains to be seen, however, whether the
law develops a jurisprudence consistent with this suggested methodology. It is hoped that the infusion of an information economics approach might trigger the switch in perspective needed in the coming policy debates to ensure the integrity of the future intellectual commons.
Appendix One
The Compilation Process

The public domain project SillyBalls is used to illustrate the steps involved in the compilation process. This program was written in 1988 and intended as an instruction tool for programmers learning the graphical user interface of the Apple Macintosh. The program does not illustrate the full features expected in a sophisticated program, but is used to illustrate the relationships discussed above between source, object and executable code. The program does nothing more than use the Macintosh Quickdraw tool to draw a series of randomly coloured and placed balls on the screen until the user clicks the mouse to end the program. The development environment used for illustration is Metrowerks Codewarrior Integrated Development Environment version 1.7.4.

The starting point is the source code files together with the project file. These are illustrated in iconographic form in Figure A1-1.

Figure A1-1 – Project Files

The project file SillyBalls.µ keeps track of the other source materials for
the program. By convention, project files are identified by adding the extension "".µ"" to the name of the file. The text based source code is located in SillyBalls.c and SillyInit.c. This "".c"" extension to the names of these files indicate that the source code is written in the C language.438 An extract of each of these C based files (without comment lines) are presented in Figure A1-2 And Figure A1-3.

**Figure A1-2 – SillyInit.c**

```c
#include "SillyBalls.h"
extern Rect windRect;
void Initialize(void)
{
    WindowPtr mainPtr;
    OSErr error;
    SysEnvRec theWorld;

    error = SysEnviron(1, &theWorld);
    if (theWorld.hasColorQD == false) {
        SysBeep(50);
        ExitToShell();
    }

    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(nil);
    InitCursor();
    GetDateTime((unsigned long*) &qd.randSeed);
    mainPtr = GetNewCWindow(rWindow, nil, (WindowPtr) -1);
    windRect = mainPtr->portRect;
    SetPort(mainPtr);
    TextSize(BobSize);
}
```

438 By convention, "".cp"" and "".p"" indicate C++ and Pascal respectively.
Appendix One – The Compilation Process

Figure A1-3 – SillyBalls.c

```c
#include "SillyBalls.h"
Rect windRect;
extern void Initialize(void);
void NewBall(void);
void MyPaint(Rect *);

void main(void)
{
    Initialize();
    do {
        NewBall();
    } while (!Button());
}

void NewBall(void)
{
    RGBColor ballColor;
    Rect ballRect;
    long int newLeft,newTop;
    ballColor.red = Random();
    ballColor.green = Random();
    ballColor.blue = Random();
    RGBForeColor (&ballColor);
    newTop = Random();
    newLeft = Random();
    newTop = ((newTop+32767) * windRect.bottom)/65536;
    newLeft = ((newLeft+32767) * windRect.right)/65536;
    SetRect(&ballRect,newLeft,newTop,newLeft+BallWidth,
        newTop+BallHeight);
    MoveTo(newLeft, newTop);
    MyPaint(&ballRect);
    MoveTo(ballRect.left + BallWidth/2 - BobSize,
        ballRect.top + BallHeight/2 + BobSize/2 -1);
    InvertColor(&ballColor);
    RGBForeColor(&ballColor);
    DrawString("\pBob");
}

void MyPaint(Rect *myRect)
{
    RGBColor myColor;
    GetForeColor(&myColor);
    InvertColor(&myColor);
    RGBForeColor(&myColor);
    PaintRect(myRect);
    InvertColor(&myColor);
}
```
The source code also includes a header file, SillyBalls.h. The ".h" extension indicates that the file contains definitions of data and prototypes of functions used by the program. An extract appears in Figure A1-4.

**Figure A1-4 – SillyBalls.h**

```c
/* Constants */
#define BallWidth 20
#define BallHeight 20
#define BobSize 8
#define rWindow 128
```

The resources for the project are contained in the file SillyBalls.m.rsrc. For this example, the resources are in a ResEdit file. ResEdit is an application in common use as a resource builder and editor. For this simple project, the only resource is the definition of the window to be used by the application. On the Macintosh, this is contained in a "WIND" resource within the SillyBalls.m.rsrc file. This is illustrated in Figure A1-5.
When CodeWarrior is used to open the project file, it displays a list of the source material to be drawn upon in the construction of the program. This is presented in Figure 5-10. Note that the project window does not contain the header file. The reason is that each of the other source files cross-reference the header file in the compiler instruction "#include SillyBalls.h", which can be found in Figures A1-2 and A1-3. This instructs the compiler, before translating source code files into binary form, to copy the header file into each of the source code files. While included in the project window, the MacOS.lib was not among the source materials presented above. This library representing basic functions is used by Macintosh systems software, and is not specific to any particular project, but instead is used by a range of projects. This library is precompiled and in a binary format, so it will not require translation, but relevant extracts are used to supplement the object code created through the other source
Once the source materials have been assembled, a compiler is used to create the object code (binaries) for each of the components of the project. The text based source code files will be sent to a C/C++ compiler for processing. The relevant elements of MacOS.lib are loaded into the project; the resource file is already in a binary form ready for linking. The compiler steps through the text based source files and creates corresponding binary object files, which are added to the project.

Following compilation, the project window in Figure A1-7 shows that the binary from compilation of SillyBalls.c and SillyInit.c are 330 and 134 bytes respectively, while 30,728 bytes of MacOS.lib has been loaded and stands ready to be linked.
The final step is the building of the software product, in this case an application or executable program. This requires a linker, which takes all the pieces of object code and arranges them in a manner compatible with the configuration of the central processing unit (CPU) of the machine on which the program is intended to run. Sophisticated linkers recognise redundant object code which is removed from the final program; such linkers may also optimise the executable code for speed in reference to a particular CPU architecture. Some other languages run an optimiser separate to the linker after the code has been run through the linker. Once this final step is complete, the output is a ready to use software product. The relevant icon for the executable application is presented in Figure A1-8.

**Figure A1-8 – Application**
Appendix Two
Commented Source Code

An example of commented source code (public domain) is presented below in Figure A2-1.

Figure A2-1 – Commented Source Code

```c
// types

typedef RectPtr *RectHandle;

/*
This is a document layout that contains the window record and references to the associating data. The window record MUST be the first field. This is because I use the window pointer returned by the Toolbox to be a pointer to my document. To confirm that the window pointer is a document pointer, I store an application reference in the window record's refCon. Then, I use a routine to test for the presence of this reference to ensure I'm looking at one of my document's windows.
*/

struct SndDocument {
    WindowRecord window;     // must be first field
    short resFile;           // document's resource file
    short vSizeNum;
    long dirID;
    ListHandle list;         // document's list of sounds
    Boolean sndInUse;        // document is using a 'snd' resource
} SndDocument;

typedef struct SndDocument *SndDocument;  // to peek at the document record

typedef struct StatusWindow StatusWindow;

typedef struct StatusWindow *StatusWindowPeeek;

typedef StatusWindow *StatWindowPeeek;

/*
This is the status window layout. The concept here is similar to the document type mentioned above. The message is a string handle used to store the current message.
*/

struct StatusWindow {
    WindowRecord window;
    StringHandle message;   // current text of status message
    long showTime;          // time window was shown
} StatusWindow;

typedef struct StatusWindow *StatusWindow;  // to peek at the document record

typedef StatusWindow *StatWindowPeeek;

/*
This is the about window layout. The concept here is similar to the document type mentioned above. The comment is a string handle used to store the current message.
*/
```
struct AboutWindow {
    WindowRecord window;
    Handle appPict;     // handle to picture of app’s name
    Handle comment;     // handle to string of about comments
};
typedef struct AboutWindow AboutWindow;
typedef AboutWindow *AboutWPtr;

// This is the template to the WIND resource. I used it to load in the WIND
// resource and then adjust the boundsRect. I also look at the procID to
// determine if it has a title bar or drag region.

struct WindowTemplate {  // template to a WIND resource
    Rect boundsRect;
    short procID;
    Boolean visible;
    Boolean filler1;
    Boolean goAwayFlag;
    Boolean filler2;
    long refCon;
    Str255 title;
};
typedef struct WindowTemplate WindowTemplate;
typedef WindowTemplate *WindowTPtr, **WindowTHandle;
Appendix Three
Decompile

Using the SillyBalls example, the decompiled version of the object code generated by SillyInit.c is set out below in figure A3-1. Note that there is little resemblance to the source code extracted in Appendix One, and even less resemblance to the full version of the source code (not reproduced) which contains extensive comments through the code, similar to Appendix Two.

Figure A3-1 – SillyInit.c (Decompiled)

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Immediate</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>02412730</td>
<td>mflr r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412734</td>
<td>stw r31,-4(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412738</td>
<td>stw r30,-8(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0241273C</td>
<td>stw r29,-12(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412740</td>
<td>stw r0,8(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412744</td>
<td>stwu sp,-96(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412748</td>
<td>addi r31,rtoct,152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0241274C</td>
<td>bl          +1380</td>
<td></td>
<td>0x02412cb0</td>
</tr>
<tr>
<td>02412750</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412754</td>
<td>sth r3,64(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412758</td>
<td>bl          +1369</td>
<td></td>
<td>0x02412cb0</td>
</tr>
<tr>
<td>0241275C</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412760</td>
<td>sth r3,66(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412764</td>
<td>bl          +1355</td>
<td></td>
<td>0x02412cb0</td>
</tr>
<tr>
<td>02412768</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0241276C</td>
<td>sth r3,68(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412770</td>
<td>addi r3,sp,64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412774</td>
<td>bl          +1364</td>
<td></td>
<td>0x02412cc8</td>
</tr>
<tr>
<td>02412778</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0241277C</td>
<td>bl          +1332</td>
<td></td>
<td>0x02412cb0</td>
</tr>
<tr>
<td>02412780</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412784</td>
<td>extsh r29,r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412788</td>
<td>bl          +1320</td>
<td></td>
<td>0x02412cb0</td>
</tr>
<tr>
<td>0241278C</td>
<td>lwz rtoct,20(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412790</td>
<td>lha r0,4(r31)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412794</td>
<td>addi r4,r29,32767</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02412798</td>
<td>extsh r30,r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0241279C</td>
<td>mullw r0,r4,r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024127A0</td>
<td>srawi r29,r0,16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024127A4</td>
<td>addze r29,r29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024127A8</td>
<td>lha r0,6(r31)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024127AC</td>
<td>addi r3,r30,32767</td>
<td></td>
<td></td>
</tr>
<tr>
<td>024127B0</td>
<td>mullw r0,r3,r0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix Three – Decompilation

024127B4: srawi r30, r0, 16
024127B8: addze r30, r30
024127BC: mli r4, r30
024127C0: mli r5, r29
024127C4: addi r3, sp, 56
024127C8: addi r6, r30, 20
024127CC: addi r7, r29, 20
024127D0: bl /*1296 */ ; 0x02412ce0
024127D4: lwz rtoc, 20(sp)
024127D8: mli r3, r30
024127DC: mli r4, r29
024127E0: bl /*1304 */ ; 0x02412cf8
024127E4: lwz rtoc, 20(sp)
024127E8: addi r3, sp, 56
024127EC: bl /*124 */ ; 0x02412868
024127F0: nop
024127F4: lha r3, 58(sp)
024127F8: lha r4, 56(sp)
024127FC: addi r3, r3, 2
02412800: addi r4, r4, 13
02412804: bl /*1288 */ ; 0x02412cf8
02412808: lwz rtoc, 20(sp)
0241280C: addi r3, sp, 64
02412810: bl /*1304 */ ; 0x02412d28
02412814: lwz rtoc, 20(sp)
02412818: addi r3, sp, 64
0241281C: bl /*1196 */ ; 0x02412cc8
02412820: lwz rtoc, 20(sp)
02412824: addi r3, rtoc, 136
02412828: bl /*1352 */ ; 0x02412d70
0241282C: lwz rtoc, 20(sp)
02412830: lwz r0, 104(sp)
02412834: addi sp, sp, 96
02412838: mtlr r0
0241283C: lwz r31, -4(sp)
02412840: lwz r30, -8(sp)
02412844: lwz r29, -12(sp)
02412848: blr
0241284C: dc.l 0x00000000 ; Invalid opcode '.....'
02412850: dc.l 0x00002041 ; Invalid opcode '....A'
02412854: lwz r0, 0(r3)
02412858: dc.l 0x0000011c ; Invalid opcode '.....'
0241285C: dc.l 0x00082e4e ; Invalid opcode '....N'
02412860: oris r23, r11, 0x4261
02412864: xoris r12, r3, 0x0

page 239
Appendix Four
Application Frameworks

Two examples of applications frameworks within the MacOS environment are EasyApp (written in C) and PowerPlant (written in C++). EasyApp is a simple yet powerful application framework written by Jim Trudeau in 1995 as a teaching device for programmers unfamiliar with the Macintosh. It provides a stand-alone shell where the basic principle is that a substantial percentage of any program behaviour can be described by some simple default functions. The source files used in EasyApp are presented in Figure A4-1.

The EasyApp section of the source materials contains the "shell" of the application and provides a range of default services to the application and its interface components. These services include: memory management (memory.c); retrieving and parsing events such as keyboard strokes and mouse actions (events.c); creation and disposal of windows (windows.c); and management of menus and menu selections (menu.c). The libraries represent the required libraries for software to run on PowerMacs (DragLib, InterfaceLib, and MWCRuntime.Lib). The programmer augments or replaces the default interface behaviour by modifying the
"hooks" to the default routines contained in the source file hooks.c, and creates customised resources by replacing the YourApp.rsrc file with appropriate new resources.

An example of a sophisticated object oriented application framework is PowerPlant, commercially available from Metrowerks and written in C++. It provides a main event loop and supporting classes which implement features such as menus, windows and printing. The source code for PowerPlant is presented in Figure A4-2.

A4-2 – PowerPlant

<table>
<thead>
<tr>
<th>File</th>
<th>Code</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MyPP_DebugHeaders.pch++</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>PP Basic Starter.cpp</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PP Basic Resource.rsrc</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>PP Basic Resource.pobj</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>PP Action Strings.rsrc</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>PP DebugAlerts.rsrc</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Commanders</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Features</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Panes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>File &amp; Stream</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Apple Events</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lists</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Support</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Utilities</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Libraries</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MacOS.lib</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPlusPlus.lib</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AEObjectSupportLib.o</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The PowerPlant environment provides a range of base classes which the programmer can draw upon to develop sophisticated applications. For example, the Pane classes handle visual elements such as windows and
pictures as well as editable text. Commander classes handle actions which involve interpreting keyboard and mouse commands (selecting Close in the File menu), broadcasting these to other objects (the relevant window) and receiving these messages and taking appropriate action (closing the window). The programmer modifies the basic framework by creating special instances (objects) of the relevant class which inherit their class's function and to which further functions can be attached. For example, to create the basic application shell, the source code will use a header file as presented in Figure A4-3.

A4-3 – PowerPlant Header

```c++
#include <Lapplication.h>
class CPPMyApp : public Lapplication {
public:
    CPPMyApp();
    virtual ~CPPMyApp();
    virtual Boolean ObeyCommand (CommandT inCommand, void* ioParam)
    virtual void FindCommandStatus(
        CommandT inCommand,
        Boolean &outEnabled,
        Boolean &outUsesMark,
        Char16 &outMark,
        Str255 outName);
protected:
    virtual void Startup();
};
```

This piece of code takes the basic application class (LApplication) and creates a specific instance or object of that class (CPPMyApp) which, along with the inherited functions outlined in the definition of the object, will form the basis for the new software development. Similar modifications can be made to program objects such as windows (from LWindow) and pulldown menus (from Lmenu).
References


References


References


References


Dean, B.V. and Goldhar, J.C. (eds), Management of Research and Innovation (North-Holland: Amsterdam).


References


References


Goldstein, P. (1994) Copyright’s Highway: From Gutenberg to the Celestial Jukebox..


Harris, J.R. (1985) "A Market-Oriented Approach to the Use of Trade Secret or Copyright Protection (or Both?) for Software" Jurimetrics Journal Vol. 25 p. 147.


References


References


References


References


References


References


References


Yen, A.C. (1991) "The Legacy of Feist: Consequences of the Weak Connection Between Copyright and the Economics of Public Goods" *Ohio
