Ekonomi och samhälle
Economics and Society

Skrifter utgivna vid Svenska handelshögskolan
Publications of the Hanken School of Economics

Nr 246

Rosa Maria Ballardini

Intellectual Property Protection for Computer Programs

Developments, Challenges, and Pressures for Change

Helsinki 2012

Intellectual Property Protection for Computer Programs:
Developments, Challenges, and Pressures for Change

Key words: computer programs, software, intellectual property law, patents, copyright, open source software

Rosa Maria Ballardini
Hanken School of Economics
Department of Accounting and Commercial Law
P.O.Box 479, 00101 Helsinki, Finland

Hanken School of Economics

ISBN 978-952-232-173-2 (printed)
ISBN 978-952-232-174-9 (PDF)
ISSN-L 0424-7256
ISSN 0424-7256 (printed)
ISSN 2242-699X (PDF)

Edita Prima Ltd, Helsinki 2012

PREFACE

While growing up in a small, remote village in the Italian Alps, I could not help but dream about exotic destinations and faraway lands. Back then I never imagined myself embarking on a journey that would eventually materialize into a PhD project at a Finnish University. However, the greatest travels are often those without clear destinations or meticulous planning, and as it turns out this has been one of my best voyages. On this journey, I've drifted and sailed. Thankfully, I've benefited from the guidance and support of many.

First and foremost, the vast academic experience and accommodating personalities of my supervisors have been crucial for developing both the research and the independent thinking skills necessary for any doctoral project. As the saying goes, "a leader is someone who demonstrates what's possible": a special thank you to Prof. Niklas Bruun, my degree supervisor, for having provided invaluable support during the whole PhD process, and for having eventually managed to convince me that actually everything is possible, if you really want it. Another heartfelt thank you goes out to Prof. Marcus Norrgård, my thesis supervisor: without your constant assistance, positive encouragement and can-do attitude this would have been a voyage with no return.

A special thank you goes to Prof. Bengt Domeij and Prof. Dan Burk, my pre-examiners: your academic work has inspired me during the whole PhD process and your constructive feedback dramatically improved the quality of my writing. I am especially grateful to Prof. Domeij, who kindly agreed to act as my opponent.

Throughout the project, the Hanken School of Economics (Department of Commercial Law) and the IPR University Center have given me physical and financial support. As such, I am extremely grateful to all the staff members working at these institutions.

Every researcher is lost without his or her peers. Throughout my journey, I was fortunate enough to be in the company of both junior and senior colleagues working in several different aspects on intellectual property. I want to thank all the members of the INNOCENT graduate school. Especially, Wim Helwegen: thank you for some of the best laughs along the road, you have been an excellent colleague and friend; thank you Liguo Zhang, Ulla-Maija Mylly, Tanja Liijestöm, Anu Pitkänen, Anette Alèn, and Amina Agovic. I would like to thank Prof. Greame Dinwoodie, who constantly took part in the activities of the INNOCENT graduate school: I am especially grateful for your empathetic encouragement and practical advices that provided invaluable support along the road.

A heartfelt thank you goes to my colleagues at Hanken, the University of Helsinki and the Helsinki Institute of Information Technology-HIIT. In particular, Pia Björkwall, Dhanay Cadillo, and Perttu Virtanen: thank you for always been there in various ways and for constantly reminding me that with the Academic freedom come great responsibilities. Also, special thanks to Nari Lee, Juha Vesala, Olli Vilanka, and Mikko Välimäki.

I spent some time on the other side of the Atlantic to complete this project. I would like to extend my gratitude to all the wonderful and helpful colleagues I had the pleasure of meeting while at UC Berkely, Boalt Hall, especially Prof. Pamela Samuelson, Colleen Chien, and Ted Sichelman: many thanks for taking the time to share your invaluable professional (and life-experience) knowledge with me.

It takes extraordinary amount of effort and a steadfast commitment for a legally trained, fundamentally technically non-adept mind to investigate and understand the intriguing complexities of the practically incomprehensible world of computer science. Fortunately, I have been able to rely on highly skilled people, excellent experts both in the field of IT and in the art of patience. Kristian Tanninen, Sami Linnanvuo, Mika Argillander, Mari Komulainen, and Olli-Pekka Piirilä: I am deeply indebted to all of you!

Thank you to all the anonymous reviewers of my academic papers, those who made me smile and those who made me cry. Thank you also to all the editors that made the outcome of this research easier for readers to digest.

You would not be reading this book if it was not for the unconditional support of my entire family, including those gone, those here, and those to come.

I would like to extend a heartfelt “grazie” to my parents, Gio Maria and Stefana, and my brother, Cirillo: you have been nothing but supportive throughout my whole life and I am forever thankful for everything you have done for me.

Pekka, my love, needless to say, you have been the truck driving this entire journey, you shaped my road better than anyone and provided the greatest support of all, steadily there for me with love, care and patience, reminding me of the much desired destination, lifting me up whenever I stumbled; Aurora, my first little angel, you cannot even properly speak yet, but already you inspire me more than words can ever say; and you, little one who has not yet had the chance to know the outside world: I would have left this road for you three if needed, but as it turned out, it is exactly you who got me to complete this voyage. There will be more to come, you will shape them all.

Helsinki, May 2012

Rosa Maria Ballardini

CONTENTS

1 INTRODUCTION TO THE STUDY....1
1.1. Aim and Research Questions....3
1.2. Structure of the Thesis....4
1.3. Research Strategy and Methodology....4
1.3.1. A Tri-Lateral Approach: History, Politics, Law, and Technology....5
1.3.1.1. The Comparative Law Approach....6
1.3.1.2. The Case Study Research Method....7

2 BACKGROUND OF THE STUDY....11
2.1. Software and its Composition....11
2.2. In Search of Adequate Legal Protection for Software....12
2.2.1. The Copyright Era....12
2.2.1.1. The United States Leads....12
2.2.1.2. Europe Follows....13
2.2.1.3. Copyright or Copywrong?....14
2.2.2. Patent Eligibility....15
2.2.2.1. The U.S. (Failed?) Experience....15
2.2.2.2. Europe: a More Restrictive Approach or a Stubborn Believer?....16
2.2.2.3. Limits of Patent Protection....19
2.2.3. FOSS: Viable Alternative or Balancing Force?....20
2.2.3.1. FOSS Licensing Model....21
2.2.3.2. Integrating the Protection Models....23

3 SUMMARY OF THE ESSAYS....24
3.1. Essay I: Scope of IP Protection for the Functional Elements of Software....24
3.1.1. Overall Objectives and Main Contributions....24
3.2. Essay II: Software Patents in Europe. The Technical Requirement Dilemma....26
3.2.1. Overall Objectives and Main Contributions....26
3.3. Essay III: The Software Patent Thicket: a Matter of Disclosure....28
3.3.1. Overall Objectives and Main Contributions....28
3.4. Essay IV: Proprietary Software vs. FOSS: Challenges with *Hybrid* Protection Models....30
3.4.1. Overall Objectives and Main Contributions....30

4 IMPLICATIONS, CONCLUSIONS AND FUTURE RESEARCH..... 33
4.1. The Software Intellectual Property Debate.....33
4.1.1. Copyright and Functions.....34
4.1.2. To Patent or Not to Patent? 37
4.1.2.1. Open Source vs. Patents?..... 40
4.1.3. Some Remarks on the Sui Generis Type of Protection 40
4.2. Avenues for Future Research 41

APPENDICES

Appendix 1 List of References..... 44

Appendix 2 Individual Essays 66

1 INTRODUCTION TO THE STUDY

This book is a study of how computer programs have challenged the thinking about and the actual use of intellectual property rights (IPRs) around the world. In general, the intellectual property (IP) system is governed by the same rules and applies equally to all fields of developments. However, the particular nature of computer software has challenged these fundamentals.

Software is a pluralistic product that contains several elements, each of which could fall into different categories of IP laws. Computer programs can be defined as "a combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions"[1]. IP protection applies differently depending on the manner in which those instructions and definitions are expressed. Currently, several protection mechanisms are available for software, including copyright, patents, trademarks, contracts, licensing agreements, and technical measures of protection. It has been suggested that none of these mechanisms, if used individually, successfully provide an adequate level of protection to computer programs.

Software technology started to appear during the 1950s, when the first computer programs were developed. As soon as it became evident that computer software was a highly complex technology that required large monetary investments, both a wide market for software and significant potential for monetary rewards were envisaged. Debates regarding the adequacy of the existing protection mechanisms for software were then initiated. It became immediately clear that each of the existing IP protection mechanisms possessed certain limitations when applied to software. These problems have generated a global debate regarding the extent to which software should be afforded IP protection. Several perspectives have been presented, and a large variety of solutions have been proposed. Some of the proposals embrace rather extremist views. For example, some suggest the complete abolition of IP protection for software in favour of an IP-free regime or the development of a completely new protection mechanism tailored to the special needs of software. In contrast, others see the IP system as playing an extremely important role in securing investments and ensuring progress in the software field and thus advocate a strong level of protection. In addition, several more balanced proposals have been launched, many of which have attempted to shape the existing IP rules such that they meet the needs of computer software. This thesis effectively contributes to this debate by providing a number of balanced, reasonable, and feasible answers to some of the major issues in the software intellectual property ecosystem. The proposed solutions were generated within the framework of the currently existing body of IP laws.

This thesis provides an in-depth analysis of some the major failures within the current European and American software IP ecosystems while focusing mainly on copyright and patent rules. Some of the problems addressed in this thesis are old problems, whereas others are new concerns generated by developments in this type of technology. Although some of these problems are as old as the adoption of IP protection for software and extensive literature has been written on these problems in the field, these issues have not been resolved. Indeed, some of the most controversial questions

[1] IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology (1990), New York, *The Institute of Electrical and Electronics Engineers*, at 66.

associated with the application of copyright and patent rules have not been answered. On these grounds, this thesis takes a step back and reconsiders the roots of this debate.

Copyright is a globally accepted mechanism for protecting computer programs.[2] Copyright can adequately protect certain aspects of software (namely, the code itself). However, the principles upon which the copyright system is based create several issues with software copyright. Among these issues, the most discussed (but unsolved) problem is most likely that copyright generally does not extend to utilitarian products and thus cannot provide protection to the software's functions (the "idea-expression" dichotomy). Other shortcomings include the contrast between the long duration of copyright protection and the short lifespan of software as well as the challenges inherent in determining the "originality" of the code. New technological developments have generated other problems, such as the difficulties created by the increased use of modularisation and object-oriented designs in computer programming[3].

With regard to copyright, this thesis focuses on the issues related to the "idea-expression" dichotomy by proposing a workable model that could be used in European jurisdictions. This book considers the related developments that have occurred in the United States. However, the decision to propose solutions for Europe was driven by the absence of a consistent interpretative framework for addressing these issues in Europe (as the relevant American jurisprudence is relatively extensive).

Overall, the discussion of the software patent framework in this thesis was triggered by the "crisis of the patent system"[4] (especially in the United States), which many claim was caused by the extension of patent protection to computer software.[5] Indeed, sings exist both to support and to defeat this claim. On the one hand, the question of patentable subject matter in computer programs continues to appear in legal disputes and has not been resolved. Some highly controversial software patents that have been granted have indicated the lack of novelty and inventiveness of these types of patents in general. The relatively low threshold for disclosure required for patents in the software field (e.g., there is no need to disclose the source code) is often controversial. Additionally, "patent thickets" are said to dominate the software patents sector. The software industry sees the patent system mostly as a cost rather than as an incentive for innovation, and companies operating in the field often claim to use their patents mainly for defensive purposes rather than to secure protection for their inventions. This information seems to suggest that the system is fundamentally flawed. However, given that the system has not yet kept progress from occurring in the field, it seems that methods of addressing these issues (regardless of their legitimacy) have emerged.

This thesis provides a richer understanding of the reasons for and consequences of extending patent protection to software. Overall, the analysis sheds light on the most challenging legal questions associated with the software patent framework and provides a solid foundation for proposing suitable solutions. With regard to patent protection, this thesis mainly proposes solutions tailored to Europe because of the background and personal interests of the researcher and because relatively few systematic studies on

---

[2] See, for examples, the WIPO Treaty on Trade Related Aspects of Intellectual Property Rights, Marocco, 15 April 1994 (TRIPs).

[3] Lipton J, "IP's Problem Child: Shifting the Paradigms for Software Protection" (2006), 58 *Hasting Law Journal* 2, at 205-251.

[4] Burk D, and Lemley M, *The Patent Crisis and How The Courts Can Solve It* (2009), The University of Chicago Press.

[5] Bessen J, and Meurer M, *Patent Failure. How Judges, Bureaucrats, and Lawyers Put Innovators at Risk* (2008), Princeton University Press.

software patent protection exist within the European legal regime. Indeed, U.S. approaches, case law and literature have been widely analysed.

Finally, the discourse on software copyright and patents is contextualised within the framework of free and open source software (FLOSS/FOSS/OSS). FOSS has become an integral part of the legal environment for software. Therefore, to provide a complete overview of the challenges associated with IP and software, we must properly consider this phenomenon. This thesis focuses on the different levels of interaction between the FOSS, copyright, and patent systems. In particular, the thesis aims to deepen our understanding of the dynamics governing the simultaneous use of the three protection mechanisms in a commercial context in order to expose the legal risks associated with this configuration and the coping mechanisms that can be implemented to navigate these risks. Thus, the purpose of this research is to generate new knowledge that can be used to strengthen our general understanding of the problems surrounding the concomitant use of FOSS, copyright, and patents in software (rather than to propose a perfect model in this respect).

Overall, this book provides a valuable contribution to the literature by revising the most relevant aspects of the IP software framework and by suggesting a number of innovative approaches to shaping the IP laws and facilitating their interpretation. The proposals included in this thesis were developed within the framework of the existing intellectual property laws in the European Union and the United States. However, the results of this book provide tools that could be used by scholars, lawyers, and policymakers around the world.

1.1. Aim and Research Questions

The primary aim of this dissertation is to increase awareness of the dynamics and controversies associated with the application of intellectual property rights (copyright and patents) to computer programs. At the centre of the study are issues related to law, technology, economics and politics.

The main research questions that will be pursued in this study include the following:

- How should the shortcomings of software copyright, particularly with respect to the "idea-expression dichotomy", be addressed by the national jurisdictions within the European Union? (Essay I)

- Should the traditional European approach to software patents, which states that computer programs "as such" do not deserve patent protection because they do not have "technical" character and do not make a "technical" contribution, be considered obsolete and thus be overruled? (Essay II)

- How could the patent law requirements be interpreted to eliminate efficiently or reduce substantially the problems associated with patent thickets in the field of software within the European system? (Essay III)

- How concrete are the legal risks associated with the simultaneous use of proprietary software (copyrighted and patented) and open source software within the same commercial software packages, and what type of coping mechanisms could proprietary software companies implement to navigate these risks? (Essay IV)

1.2. Structure of the Thesis

This thesis comprises an introductory chapter and four original essays. This first chapter is divided into three main parts. The first part, "Introduction", presents the motivations, research objectives, research strategy, research methodology, and overall structure of the thesis. The second part presents the general background for the research. Specifically, the second part works with the definition of "software" and from a historical and political perspective on how the IP system has evolved to adapt to the challenges posed by software technology. The third part summarises the content of the essays and highlights their individual contributions. Part four summarises the discussion, conclusions, and limitations of the study. This section synthesises the results and presents the overall contributions of the book, its academic and practical implications, and its limitations and avenues for future research. Because part three simply provides a summary of the essays within the dissertation, a reader who is already familiar with the articles could read only parts one, two and four while skipping part three.

The essays cover a relatively wide range of themes, the common denominator of which is the process of adapting intellectual property rights (mainly patents and copyrights) to computer software. The first essay ("Scope of IP Protection for the Functional Elements of Software") addresses the problems associated with intellectual property protection for the functional elements of software, with a specific focus on the applications of program codes. The second essay ("Software Patents in Europe. The Technical Requirement Dilemma") analyses the doctrinal issue of the "technical" criterion as it applies to computer-implemented invention (CII) patents in Europe. The main focus is the current EPO approach. This essay asks whether the EPO approach is too rigid and obsolete and should thus be abandoned. The third essay ("The Software Patent Thicket: a Matter of Disclosure") investigates both the reasons for and the consequences of the emergence of patent thickets in software. The essay then suggests proper ways to navigate these thickets. The fourth essay ("Proprietary Software vs. FOSS: Challenges with *Hybrid* Protection Models") investigates the reasons for and the consequences of the fact that open source software has now begun to be used by proprietary companies. The aim is to throw light on both the legal challenges associated with the model and the coping mechanisms that can be used to navigate these risks.

The first two essays are peer-reviewed articles published in international legal journals, the third is a peer-reviewed contribution to a book, and the fourth is a peer reviewed article published in an electronic publishing series.

1.3. Research Strategy and Methodology

This study investigates different aspects of the functions and uses of the intellectual property system within the software industry. The project is interdisciplinary and is based on prior legal, historical, political, economics, and technological studies. The central methods used in this thesis include legal, historical and technology-based approaches. The tools used to conduct this tri-lateral method-based study were the comparative law method and the case study research method.

1.3.1. A Tri-Lateral Approach: History, Politics, Law, and Technology

The study is multidisciplinary, as is the method used. The study builds on a historical, political, legal, and technical analysis.

The study uses legal theory to analyse the current regulatory framework governing the activities in the field of intellectual property law and software. Traditionally, general legal doctrine describes and systematises legal sources and legal arguments.[6] Furthermore, general legal doctrine produces self-reflection about the goals it serves and the methods it uses. It also includes philosophical insights and tools.[7] Legal doctrine is usually neither purely descriptive nor purely normative. Under general legal doctrine, in fact, statements about valid law can be understood both as descriptions of existing law and as normative recommendations regarding improvements to legal rules.[8]

This study focuses on the systematisation and interpretation of European laws by using the American legal system as a point of comparison. The main focus is the regional and international agreements within the area of software IP protection. In the field of copyright law, the study relies primarily on the EU Software Copyright Directive[9] and the U.S. Copyright Act[10], whereas in the field of patents, the European Patent Convention (EPC)[11] and the U.S. Patent Act[12] are mostly considered. The study also considers national legislation and case law. However, these laws serve primarily as examples of how the issues at stake could or have been addressed.

Although useful, the traditional normative legal approach does not provide as comprehensive a perspective as this multidisciplinary study requires. Accordingly, the normative approach is integrated with historical, political, legal, and technological approaches. The historical and political material presented in the introductory portion of this thesis sheds light on the developments that have occurred both in the field of software IP law and in software technology *per se*. The different levels of intellectual property mechanisms utilised for computer programs have evolved in accordance with the new technological developments in this area. The connections between the past historical, legal, and technological advancements in the software industry provide a foundation for our understanding of the current legal framework in this area. Furthermore, it was necessary to consider this material to propose suitable alternatives to the research problems identified in this thesis. All three of these considerations needed to be discussed to present a comprehensive, practical and valuable analysis.

The economic perspective is essential to this type of research. However, this study does not employ a traditional law and economics approach, where concepts from economics

---

[6] Klami H, *Comparative Law and Legal Concepts, The Method and Limits of Comparative Law and its Connection with Legal Theory* (1981), Turku, Finland: Vammala, at 93.
[7] *Ibid.*
[8] Egg S, *Analysis of Dis/Agreement with Particular Reference to Law and Legal Theory*, (First eds., 2003), Kluwer Academic Publishing, at 312-351.
[9] Council Directive of 14 May 1991 on the legal protection of computer programs (91/250/EEC) (Software Copyright Directive).
[10] 17 U.S.C. (2000). In September 16th, 2011 a new patent act was passed in the United States, the Leahy-Smith America Invents Act (AIA). The timeframe during which this PhD project was conducted did not allow considering the principles included in the AIA. However, the changes introduced by such an Act do not disturb the overall analysis, or the conclusions of this manuscript as a whole.
[11] Convention on the Grant of European Patents of 05 October 1973 (European Patent Convention).
[12] 35 U.S.C..

are used to explain the effects of the law, to assess which legal rules are economically efficient and to predict which legal rules will be promulgated[13]. Instead, this study utilises economic analysis to enhance the understanding of particular legal failures. Qualitative "explorative" case study research is presented in Essay IV to generate a new approach to using legal protection mechanisms (especially copyright, patents, and open source) with computer programs from a business perspective. Case study research was especially helpful in this area because the "boundaries" of the researched phenomenon (i.e., the possible problems that might result from the simultaneous use of proprietary and open source software) are not clearly evident.[14]

1.3.1.1. The Comparative Law Approach

The thesis aims to highlight some of the most challenging legal questions associated with the intellectual property protection of computer programs. More specifically, it proposes a series of innovative, more workable solutions to the challenges that exist in the European context.

All of the essays included in this thesis utilise the comparative method of analysis, though in different ways. Generally, comparison is used in these essays to identify the common "core" that binds the research questions together and to shed light on the approaches used in different jurisdictions to solve the problems at hand. Therefore, all of the essays begin by highlighting the substantial similarities and differences between the national, regional, and international approaches presented in this thesis. Generally, the selected national and regional cases are treated as examples of solutions to specific problems related to software IP protection. These cases constitute the basis of comparison for the subject matter at hand. Furthermore, it is essential to consider the conclusions of the courts to highlight the limitations within the current legal framework for software IP protection and to find suitable solutions to these issues within European law.

In Essays I, III and IV, the two levels of comparison consider the approaches followed in the United States with the ones developed in Europe. In particular, the generally more developed American perspective on the matters constitutes a solid background for discussing and proposing workable solutions to the problems in the European framework. Essay I includes a further level of comparison in the European discussion, as the case law developed in three European countries, the United Kingdom, Germany, and France, is considered. In Essay II, the two levels of comparison are considered only at the European level. Specifically, the "core" of the problem is addressed under the terms of the case law developed at the European Patent Office (EPO), whereas the different approaches followed by the national courts of the United Kingdom, Germany, and France provide the second level of comparison. Essays I, II, and III utilise the comparative method to formulate proper solutions to the problems at the European level, whereas Essay IV utilises the element of comparison to provide a relevant legal context to the problems discussed in the paper.

---

[13] See, for instance, Epstein R, "Law and Economics: Its Glorious Past and Cloudy Future" (1997), *University of Chicago Law Review*, at 1167-1174. For information on law and economics and IP law, specifically, see Landes W, and Posner R, *The Economics Structure of Intellectual Property Law* (2003), Belknap Press of Harvard University Press; and Posner R, *Frontiers of Legal Theory* (2004), Harvard University Press.

[14] Yin R, *Case Study Research: Design and Methods*, Volume 5 of Applied social research methods series (Forth eds., 2008), Sage Publications, Inc..

As previously mentioned, two of the essays also include a national element of comparison within the European framework, as the British, German, and French approaches are considered. These countries were selected because they are representative of the types of legal regimes that exist in Europe (common law systems and civil and Germanic law systems) and also because of their extensive jurisprudence on the application of copyright and patents to computer programs.

The American legal system was chosen as a point of comparison because of its historical connection to the legal, technical and economic developments in computer software. Indeed, the U.S. has led the development of both IP law and software technology. Most of the relevant controversies and debates have originated in the United States.

There are limits to the selected method. Traditionally, the comparative method is used to compare legal systems or the substantive laws in a specific field.[15] In other words, the aim of traditional comparative research is to learn from another system and to transplant functioning solutions from one system to another.[16] However, this thesis utilises the comparative method differently. This research does not attempt to apply the lessons of one specific jurisdiction to another. Instead, the goal of this thesis is to learn from the combined knowledge of all of the national, regional, and international legal systems mentioned rather than to identify one specific system as superior. Therefore, this thesis compares the solutions adopted in the selected countries to provide IP protection for computer programs. Additionally, this study deeply analyses the limitations of these proposals and aims to identify more workable alternatives. Although the solutions formulated in this thesis are tailored to European copyright and patent systems, the chosen method makes it necessary to discuss these issues on an abstract level as well. Furthermore, the global nature of software and the problems addressed in the research suggest that the conclusions and solutions proposed will also be relevant to other (Western) legal regimes.

1.3.1.2. The Case Study Research Method

Some General Background

This thesis also utilises case study research. More specifically, case study research is implemented in the fourth essay.

Several definitions of case study research exist, each of which reveals something about case studies and contributes to our general understanding of the nature of this type of research. For instance, Robert Yin defines a case study as "An empirical enquiry that investigates a contemporary phenomenon within its real life context, especially when the boundaries between phenomenon and context are not clear evident"[17], whereas the definition presented by Sharan Merriam suggests that "A qualitative case study is an intensive, holistic description and analysis of a single instance, phenomenon or social unit".[18] Generally, case study research is a method of enquiry employed with existing

[15] Örucu E, "Developing Comparative Law", in Örucu E, and Nelken D, *Comparative Law a Handbook*, (2007), Hart Publishing, at 43-66.
[16] Zweigert K, and Kötz H, *An Introduction to Comparative Law* (Third eds., 1998), Oxford University Press, at 1-13.
[17] See note 14 above.
[18] Merriam S, *Qualitative Research and Case Study Applications in Education* (1998), San Francisco: Jossey-Bass Publishers, at 21.

phenomena within a "real-life" context, especially if the boundaries among these phenomena are not evident.[19]

Fundamentally, the case study methodology employs a "qualitative" paradigm rather than a "quantitative" one. In fact, case studies can be conceived of as satisfying the three pillars of the qualitative method: describing, understanding and explaining. Thus, the aim of case studies is to establish meaning rather than location. Generally, qualitative research is "inductive", i.e. it is more discovery- and process- driven. It is also "naturalistic" (i.e., suitable for real-life situations and contemporary events).[20]

The literature identifies several types of case studies. For instance, Yin identifies three types of case studies: "exploratory" studies, which are generally used as pilot projects and as preliminary studies; "explanatory" studies, which are often used for casual research (e.g., to investigate cause-effect relationships); and "descriptive" studies, which develop a descriptive theory of the phenomenon under investigation, usually within its context.[21] Other types of case studies have been identified by several authors. For example, Stake mentions "intrinsic" case studies (in which the researcher has an interest in the case), "instrumental" case studies (in which the study is conducted to understand a phenomenon that extends beyond the actual case), and "collective" case studies (in which a group of cases or objects is studied).[22]

As in all types of research, internal validity, external validity, and reliability must be ensured. According to Yin, "construct validity" can be achieved by using multiple sources of evidence. More specifically, Yin identifies six primary sources of evidence: documentation (e.g., letters, memoranda, agendas, and study reports), archival records (e.g., service records, maps, charts, lists of names, survey data, and personal records), direct observations (i.e., if the investigator makes a site visit to gather data), interviews (open-ended, focused, or structured), participant observations (i.e., if the researcher participates in the event being studied), and physical artefacts (i.e., physical evidence that might be gathered during a visit).[23] The "external validity" of case study research is often a subject of debate because a single case (or a limited number of cases) cannot be used to draw general conclusions. Yin suggests that external validity can be achieved from theoretical relationships. Furthermore, the goal of the research should establish the parameters of the study: if the study meets those established objectives, even a single case could be considered to provide external validity. Finally, the "reliability" of the study is sought by using the "case study protocol", a document that the investigator should draft to outline the procedures and general rules that will be employed in the case study. The reliability and accuracy of a study will be enhanced if the draft report is reviewed not only by the researcher's peers but also by all of the participants and informants involved in the case. [24]

Generally, in a case study, it is most desirable to disclose both the case and the individuals. However, anonymity may be necessary on some occasions. For instance, if the study addresses a controversial topic, if issuing the case study report might affect the actions of those who were studied, or if disclosing the identities of the individuals involved is unnecessary because the study aims to define an "ideal type", then anonymity is needed. However, if anonymity is justified, other compromises must be

---

[19] See note 14 above, at 3-25.
[20] *Ibid.*
[21] *Ibid.*
[22] Stake R, *The Art of Case Research* (First eds., 1995), Sage Publications, Inc..
[23] *Ibid.*
[24] See note 14 above, at 127-164.

sought (e.g., the study may maintain the anonymity of only the individuals but not of the cases or may name the individuals but not attribute any specific opinions to them). Another option might be to avoid composing a single-case report and to report only a cross-case analysis instead (although this option is valid with only multiple case studies). [25]

The Specific Application

In the fourth essay in this thesis, a case study analysis was conducted. As explained later in detail, this paper investigated the reasons why open source software is now often used by proprietary companies and the consequences of this fact. The essay specifically focuses on the problems that proprietary companies may encounter if they incorporate both open source and IP protected code in the final proprietary software that they release to the market (under the *hybrid* protection model). Case study research was utilised to enhance understanding of the most concrete legal risks of *hybrid* models. Toward this end, the case study used qualitative analysis. Using quantitative techniques might have obscured some important information, including whether the *hybrid* protection method is efficient and what type of specific problems it can generate in practice. The types of research questions to be investigated justified the use of "intrinsic" and "collective" case studies.

The subject of the study was identified by representative companies operating in the software field. A multiple case study was conducted because more than one case was available for replication; several companies use the *hybrid* model and might have encountered the problems that were the object of the study.[26] To select the subjects of investigation, the study used an information-oriented technique rather than random sampling.[27] To maximise what could be learned in the period of time available for the study, the study chose the companies based on their representativeness given the overall research objective (i.e., they were "key" cases). The study considered six cases, including one consultancy company and five software companies.

The study used two different sources of evidence: documents and interviews. The documents mainly included academic studies (legal and economics-oriented studies), case law, legislation, publicly available companies' policies regarding intellectual property and open source software, company websites, and newspaper articles.

Interviews were the most important source of information in the study, and open-ended interviews were used.[28] For instance, by interviewing key individuals, the study was able to acquire information that it might not have collected using a questionnaire. The key respondents were asked to comment on the research questions not only based on the perspective of their own company but also (and more importantly) based on their extensive knowledge of the field. The respondents were free to propose solutions or provide insights regarding the subject matter and were also asked to comment on evidence obtained from other sources. This "open" method expanded the depth of the relevant data gathered.

---

[25] *Ibid*, at 164-190.

[26] For more information on case study research, see Yin R, note 14 above; Yin R, *Applications of Case Study Research* (Third eds., 2011), Sage Publications, Inc..; and Stake R, note 22 above. See also Tellis W, "Introduction to Case Study" (1997), 3 *The Qualitative Report* 2, at: http://www.nova.edu/ssss/QR/QR3-2/tellis1.html; and Tellis W, "Application of a Case Study Methodology" (1997), 3 *The Qualitative Report* 3, at: http://www.nova.edu/ssss/QR/QR3-3/tellis2.html.

[27] *Ibid*.

[28] *Ibid*.

Based on both the documents consulted and the answers received during the interviews, a draft report was written.[29] This report was then reviewed by all of the study participants, who verified the accuracy of the answers and the overall conclusions and observations. The draft report was also discussed with several external peers, who critically challenged the results and provided relevant feedback. This process enhanced the accuracy of the case study.

The anonymity of the interviewees and their respective companies were preserved because the issue is controversial and the collected information was considered confidential by the interviewed companies. Thus, a cross-case analysis was used instead of a single-case report.[30] The case study report synthesised the lessons learned from all of the companies, and examples from the cases were discussed in the different sections ("Compliance and compatibility", "Reciprocity of FOSS licenses", "Ownership of rights", "Patent infringement and litigation", "Dilution of the company's own patents", and "Lack of warranties and indemnifications").

The case study was relevant because it provided in-depth answers to the theoretical issues formulated in the first part of the paper. The empirical analyses shed light on the most concrete legal risks associated with *hybrid* models and on the coping mechanisms that can be used in practice to navigate such challenges.

[29] See note 14 above at 141-167.
[30] *Ibid*, at 170-173.

2 BACKGROUND OF THE STUDY

2.1. Software and its Composition

What exactly do we mean by "software"? The significance of words such as "computer program", "algorithm", "source code", and "object code" is not a necessary subject of debate in everyday software engineering.[31] However, if they become a part of legislation, the need for a precise definition of these terms arises.

Although no European legislation defines "software", computer science and software engineering have formulated hundreds of quasi-official definitions. For example, the IEEE Standard Glossary of Software Engineering Terminology states that software includes:

> "Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system".[32]

It follows that a "computer program" is simply a part of software and that patenting a computer program is not the same as patenting software. It is commonly understood that software includes both the computer program and its ancillary materials (e.g., documentation and computer files or data).

Computer programs *per se* can be defined as "a combination of computer instructions and data definitions that enable computer hardware to perform computation or control functions".[33] More specifically, computer programs compose of both source code and object code. The source code embeds the programmer's instructions in a programming language. The source code usually resembles English and is the only part of the program that is readable by humans. However, for the hardware to understand the instructions provided in the source code, the source code must be translated into machine-understandable instructions, either by a program called a "compiler", or by an "interpreter". The compiler produces an object code version of the program in a machine-readable language, while the interpreter translates the source code instruction by instruction while the program is running. The object code (or binary code) is usually a series of bits ("zeros and ones") and thus cannot be read by humans. Indeed, not always software is compiled, i.e. not always source and object code appear to be distinct. Notably, as mentioned, in software that is interpreted rather than compiled, such a distinction is not possible to make. However, it should be noticed that, although compiling and interpreting are the two principle means of implementing programming languages, they are not necessarily always mutually exclusive categories. In fact, most interpreting systems perform also translation works in the same way as compiler systems.

This short description of the composition of computer programs sheds light on the characteristics of these programs that must be considered when discussing intellectual property protection.

First, computer programs are pluralistic. They possess several elements, each of which could be protected under different categories of IP law. Computer programs are both

---

[31] Sarvas R, "A software engineering view to the decisions of the European Patent Office concerning computer programs" (2001), *Helsinki University of Technology*, at 11-16.
[32] See note 1 above, at 66.
[33] *Ibid*, at 19.

literary and functional works. The program's source code is expressed in writing and thus renders the program a literary work of art. At the same time, however, once the code is processed by a machine and has ordered the hardware to perform certain functions, the code can be viewed as utilitarian.

Second, software technology is one of the fastest growing and most complex industries because of the process by which the products are built. The computer software industry is an incremental industry in which new developments must build on existing technologies. Accordingly, innovation occurs through small improvements, and products incorporate many different components rather than single inventions. These components can each be protected by IPRs.[34]

All of these concerns must be carefully investigated to provide a comprehensive analysis of how intellectual property applies to and interacts with computer programs.

2.2. In Search of Adequate Legal Protection for Software

This section presents the history of intellectual property protection for software and the political and legal debates surrounding the application of IPRs to computer programs. A solid background in the evolution of the application of intellectual property rules to software makes it possible to address the research questions subsequently investigated in the essays.

This section (together with section 4) also discusses the most recent developments in the jurisprudence and the legislation related to software IP. Some of these developments occurred after the publication of the essays included in this doctoral dissertation and have consequently not been discussed in the papers. However, these developments do not affect the research results presented in the essays or the proposed solutions.

2.2.1. The Copyright Era

2.2.1.1. The United States Leads

The debates regarding the forms of protection afforded to computer software date back to the 1950s and 1960s, when the first computer programs were developed. The debates originated in the United States. Initially, IPRs and proprietary rights were not generally a concern for computer programmers. Most programs were developed at universities or research centres that encouraged open exchange, with researchers building programs by sharing source codes and related modifications. Generally, in this period, the software was bundled with the hardware rather than sold individually. The few computer programs that were sold during this initial phase were usually licensed under restrictive trade secret agreements.[35]

---

[34] Lemley M, and Shapiro C, "Patent Holdup and Royalty Stacking" (2007), 85 *Texas L. Rev*. 7, at 1991-2050.

[35] Samuelson P, "A Case Study on Computer Programs", in *Global Dimension of Intellectual Property Rights in Science and Technology* (1993), Office of International Affairs, National Research Council, NATIONAL ACADEMY PRESS Washington D.C., at 284-318.

As soon as programs became more complex and large monetary investments became necessary, the discussion regarding more adequate protection mechanisms for software began. The first suggestions regarding the application of copyright law to computer programs were developed in the United States. In 1964, the US Copyright Office started to accept computer programs for copyright registration. Initially, the office required the full source code to be filed, which would be made available to the public.[36] Furthermore, the office accepted programs under the "rule of doubt" based on the concern that software was both text (and thus copyrightable) and mechanical (i.e., functional).[37] In addition, software was not yet a mass-marketed product. For these reasons, the number of software-related applications filed with the copyright office was limited during the 1960s and 1970s. Consequently, trade secrecy and licensing agreements were still the most commonly used protection mechanisms for computer programs.[38] Furthermore, there was still an open system for sharing ideas related to program development, and the results were often published. Although a number of commercial products began to use software, the commercial ventures usually arose after the research results were published. Thus, these ventures did not create impediments for computer scientists.[39]

Important amendments that were adopted under the recommendation of the National Commission of New Technological Uses and Copyrighted Works (CONTU) during the 1980s defined the scope of copyright protection for computer programs. CONTU saw copyright as a valuable protection mechanism for computer programs. Accordingly, the U.S. Copyright Act was amended to include software as officially copyrightable. Additionally, in this same period, the Copyright Office decided to drop the requirement that programs provide the full text of their source code. Together with the advent of the first personal computers (PCs) and the mass marketing of software, these legal developments positioned copyright as the primary legal instrument of protection for computer programs in the U.S.

2.2.1.2. Europe Follows

The EU adopted copyright protection for software slightly later than America. Initially, the discussions focused on whether copyright law was consistent with the Treaty of Rome. After the Court of Justice declared that copyright fell under Art. 36 of the Treaty[40], software copyright became a subject of discussion during the efforts to develop an internal market. These initiatives generated the 1988 Green Paper on Copyright and the Challenge of New Technology. After long and intense internal and public pre-legislative debates, the European Software Copyright Directive was finally issued in 1991.

Prior to the adoption of the 1991 Directive, copyright (or *droit d'auteur*) was generally accepted as a protection mechanism for computer programs. However, the Member States interpreted the concept of software copyright in different ways. As it became

---

[36] Samuelson P, "CONTU Revisited: The case against copyright protection for computer programs in machine-readable form" (1984), *Duke L.J.* 4, at 663.

[37] The Office based its doubts on the Supreme Court decision on *White-Smith Music Co. vs. Apollo* (1908), 209 S.Ct. 1.

[38] See note 35 above.

[39] *Ibid.*

[40] For the cases in which the Court of Justice declared that copyright was covered by the Treaty of Rome, see, for example, C- 78/70 *Deutsche Grammophon* [1971] E.C.R. 1-487; and Case C-62/79 *Coditel vs. Cine Vog Films* [1980] E.C.R. 1-881.

clear that software would be highly important to commerce both in the EU and around the world, it seemed important to remove any possible barriers to the free circulation of computer programs within the Community. Copyright was chosen as a protection mechanism not only because most of the EU Member States and other countries (e.g., the U.S.) had already adopted it but also because other mechanisms, such as patents and contracts, were felt to provide insufficient or inadequate protection for software. The American acceptance of software copyright influenced the European Community's decision.

The draft Directive was the subject of intense debate within the Community and of strong lobbying by several firms, many of which were based in the U.S.. There were two main issues: the decompilation of code and the protection of the interfaces used in internal programs. A balanced compromise was reached: decompilation was allowed only if it was necessary to achieve interoperability with independently created programs, and the Directive excluded the "ideas" and "principles" associated with the programs but not the algorithms, interfaces, or program logic. Only the ideas associated with these categories were excluded. Therefore, much was left to the individual States' interpretations.

2.2.1.3. Copyright or Copywrong?

Doubts regarding the ability of copyright law to protect computer programs were raised during the initial discussions concerning the question of whether to apply these protection mechanisms. For instance, as previously mentioned, the U.S. Copyright Office initially accepted software copyright only under the "rule of doubt". Although a long time has passed, some of these questions remain unresolved.

There are some uncontroversial aspects of software copyright. For instance, it is widely accepted that copyright protects against the exact (or nearly exact) copying of program code (i.e. literal infringement). Furthermore, it protects the aspects of a program that can be clearly identified as "expressive", such as certain aspects of user interfaces (e.g., videogame graphics).[41]

However, substantial controversy remains regarding the ability of copyright law to protect other aspects of software. Some of these problems have been solved via legislation and its interpretation. For instance, the problem of how to apply the criterion of "originality" to computer programs has been partially solved by deciding that the quantitative or aesthetic merits of computer programs should not be relevant to the question of whether they have copyright protection.[42] Other issues persist. Most of these questions centre on whether and to what extend the non-literal elements of programs are copyrightable. These problems will be detailed and contextualised based on the results presented in section 4.1.1.

---

[41] See note 35 above.

[42] See 17 U.S.C. § 102 (b). See also Preamble of the Software Copyright Directive: "Whereas, in respect of the criteria to be applied in determining whether or not a computer program is an original work, no tests as to the qualitative or aesthetic merits of the program should be applied".

2.2.2. Patent Eligibility

2.2.2.1. The U.S. (Failed?) Experience

The most controversial debates related to software and patents have centred on the issue of patentable subject matter. United States law does not explicitly exclude computer programs from the category of patentable subject matter. Instead, if a program meets the requirements of novelty, non-obviousness, and usefulness (and enablement), the program can be patented.[43] The patentable subject matter requirement under U.S. Code 35 § 101 is considered a rather flexible concept that has been interpreted differently in accordance with economic and technological changes.

Historically, computer programs were not eligible for patent protection in the United States. During the 1960s and 1970s, when the debate originated, computer programs were denied patent eligibility primarily because they were merely mathematical methods. They were equated with mental processes or to steps that could be performed by the human mind, and the simple fact that such processes were executed in a machine (a computer) did not change their basic nature.[44] This initial policy was also based on the report of a presidential commission that had indicated how the office could cope in an "area of exploding technology".[45] Thus, at the time, inventors did not commonly seek patent protection for programs, especially after the Supreme Court passed decisions indicating that software was ineligible for patents.[46]

As computer programming rapidly advanced, companies began demanding stronger legal mechanisms to prevent their competitors from "cloning" their programs. In this way, companies promoted innovation in the field.[47] Within these debates, the need for patent protection was communicated. During this period, theories supporting the eligibility of software for patents were developed. These theories were based on the idea that even if a program contains a mathematical formula, if the formula is implemented within a structure or via a process that "when considered as a whole is performing a function which the patent laws were designed to protect", then the program should be patentable.[48] This interpretation opened the door to some software patents. However, during this period, neither mathematical algorithms nor business methods were considered eligible for patent protection.

These last restrictions were eliminated in 1998, when a Federal Circuit decision in the State Street Bank case[49] affirmed what became known as the "useful, concrete, and tangible result" test. In this approach, processes can be patented if they produce a "useful, concrete, and tangible result", even if the results are expressed "in numbers, such as price, profit, percentage, cost, or loss". One consequence of the State Street decision was the creation of the new class 705 in the U.S. patents classification system for "Data processing: financial, business practice, management, or cost/price

---

[43] See 35 U.S.C. § 101, 102, and 103.

[44] This was the so-called "Mental Step Doctrine". See *In Re* Abrams (1951), Fed. Cir. 188 F.2d 165, 89 U.S.P.Q. (BNA) 266.

[45] Report of the President's Commission on the Patent System, "To Promote the Progress [...] of the Useful Arts", in an Age of Exploding Technology, 13 (1966).

[46] See *Gottschalk vs. Benson* (1972), 409 S.Ct. 63, and *Parker vs. Flook* (1978), 437 S.Ct. 584.

[47] See note 35 above.

[48] *Diamond vs. Diher* (1981), 450 S.Ct. 175.

[49] *State Street Bank & Trust Co. vs. Signature Financial Group Inc.* (1998), Fed. Cir., No.: 149 F.3d 1368.

determination". The patent office was inundated with claims that did not involve technology, including claims related to finance, arbitration, and teaching.

The Bilski case was essential to the interpretation of patentable subject matter under 35 U.S.C. § 101. Although the dispute involved a pure business method, because business methods are often implemented using computer programs, software inventions have also been actively considered in the Bilski debates. After a rejection from the U.S. Patent Office (PTO) and the Board of Patents Appeal and Interfaces (BPAI)[50], the case went to the Court of Appeal for the Federal Circuit (CAFC), which issued an *en banc* decision on October 30, 2008 that substantially reconsidered the criteria for assessing the patentability of claims. In *In Re* Bilski[51], the CAFC suggested that the "useful, concrete, and tangible result" test was "insufficient to determine whether a claim is patent-eligible under § 101" and adopted the "machine or transformation" (MoT) test. Accordingly, patents can be granted if the process is connected to a specific machine or transforms a particular article into a different state or thing. However, the use of a specific machine must impose meaningful limitations on the scope of the claim for the process to be eligible. Finally, Bilski appealed to the Supreme Court by filing for a writ of *certiorari*. The Court accepted and issued its decision on June 28, 2010.[52] The Supreme Court confirmed that categorical exclusions (e.g., for business methods) are not legally acceptable under U.S. patent law. The Court revised the CAFC's decision and called the "machine or transformation" test a "useful and important clue" but not the "sole" test of process patentability. Therefore, under the current rules, the State Street and the MoT tests are both used to determine patentability in the United States.

It is worth mentioning that the Federal Circuit has recently petitioned for the rehearing *en banc* of a case concerning patentable subject matter under 35 U.S.C. § 101. In *Ultramercial vs. Hulu*, a district court initially rejected a software-related patent claim because the invention was deemed an unpatentable abstract idea. However, the CAFC subsequently ruled that because the invention was "a practical application" and that "viewing the subject matter as a whole, the invention involves an extensive computer interface", the invention was patentable.[53] The *en banc* petition filed by one of the alleged infringers was also supported by the Electronic Frontier Foundation (EFF) for the following reasons: 1) the CAFC failed to follow the Bilski rules; 2) the court's recent decisions (e.g., *Classen Immunotherapies vs. Biogen IDEC*[54] and *CyberSource Corp. vs. Retail Decisions, Inc.*[55]) used inconsistent approaches in applying the law; and 3) there is a growing intra-circuit division regarding patentable subject matter. Patentable subject matter is also at the centre of multiple pending cases at the Supreme Court, such as the recently decided *Mayo vs. Prometheus*[56], a dispute regarding patents and isolated human DNA.

2.2.2.2. Europe: a More Restrictive Approach or a Stubborn Believer?

Under the European system, the discourse regarding patents and computer refers to computer-implemented inventions (CII), whereas terms such as "software patents" are considered inaccurate. CII are defined as:

---

[50] *Ex Parte* Bernard L. Bilski and Rand A. Warsaw (2006), BPAI, No.: WL 4080055 at 1.
[51] *In Re* Bilski (2008), Fed. Cir. F.3d, No:. WL 4757110, 88 U.S.P.Q.2d (BNA) 1385.
[52] *Bilski vs. Kappos* (2010), 08 S.Ct. 964.
[53] *Ultramercial, LLC vs. Hulu, LLC.* (2011), Fed. Cir., No.: 09-cv-6918.
[54] *Classen Immunotherapies vs. Biogen IDEC* (2011), No.: 06-1634-1649.
[55] *CyberSource Corp. vs. Retail Decisions, Inc.* (2011), Fed. Cir., No:. 04-cv-03268.
[56] *Mayo vs. Prometheus* (2011), 10 S.Ct. 1150.

> "Inventions whose implementation involves the use of a computer, computer network, or other programmable apparatus"[57].

In Europe, patent eligibility is governed by Articles 52, 56, and 83 of the European Patent Convention, which list the following patentability requirements: invention, novelty, an inventive step, industrial applicability, and sufficient disclosure. In the context of computer-implemented inventions, the most discussed requirement is "invention". This requirement is specifically referred to in Article 52(2), which excludes computer programs (and methods of doing business) "as such" from the category of "inventions" and thus from the category of patentable subject matter.[58] Although the term "invention" is not defined under the EPC, it has been inferred in practice that inventions must be "technical" in nature.[59] Accordingly, computer programs (such as the other excluded categories) are considered as "non-technical".

The debate over the eligibility of computer programs for patents began before the EPC was signed in 1973[60] and continued afterwards, as there was no unified approach to assessing the patent eligibility of programs.[61] However, the interpretation of the phrases "as such" and "technical" has evolved across the years. Indeed, the original strict interpretation of "as such", according to which most programs were barred from the patent sphere, has been expanded.

The basis of the EPO perspective on patents for computer-implemented inventions (CII) is the Vicom decision of 1986[62]. This decision stated that even if a process is based on an algorithm (and thus made of non-patentable elements), it can be considered both an invention and patentable as long as "a technical contribution is made to the known arts" (in the "technical contribution" approach). In the nineties, two cases involving IBM determined that programs are eligible for patent protection if they are technical in nature[63]. Specifically, the IBM rulings found that even though the mere interaction between a program and a machine did not render it technical, this requirement was fulfilled "in the further effect deriving from the execution (by the hardware) of the instructions given by the computer program. Where said further effects have a technical character [...] an invention [...] can be [...] the subject-matter of a patent" (under the "technical effect" or "further technical effect" approach). The trend that narrowed the meaning of the term "as such" continued under the "any hardware" approach. This approach was originally developed in Pension Benefit, a case that addressed the question of business methods[64] and was further elaborated in two computer program-related cases, Hitachi[65] and Microsoft[66]. Under the "any hardware" approach, any software embedded in hardware (however mundane) has "technical

---

[57] See "Patents for Software? European Law and Practice" (2008), *European Patent Office*.

[58] See EPC, Article 52(2)(3).

[59] The concept of "technical character", which originates from German tradition, was officially adopted under European patent law in 1984 with the issuing of the EPO Guidelines for Examinations.

[60] Meshbesher T, "The Role of History in Comparative Patent Law" (1996), 78 *J. Pat. & Trademark Off. Soc'y* 594. See also Cohen D, "Comment Article 69 and European Patent Integration" (1998), 92 *Nw. U. L. Rev.* 1082, at 1092-1112.

[61] Kikuchi M, "Patent eligibility and patentability of computer software patents in the United States, Europe and Japan" (2009), 16 *CASRIP Newsletter* 3.

[62] T 0208/84 Computer Related Invention/Vicom [1987] OJEPO 14.

[63] T1173/97 Computer Programs Product/IBM [1999] OJEPO 609; T 0935/97 Computer Program Product/IBM [1999] OJEPO 609.

[64] T 0931/195 Controlling Pension Benefit System/PBS Partnership [2001] OJEPO 441.

[65] T 0258/03 Auction Method/Hitachi [2004] OJEPO 575.

[66] T 0424/03 Microsoft/Data transfer expanded clipboard formats [2006].

character". The focus should be on whether the supposed invention is actually inventive with respect to the prior arts. The "any hardware" approach clearly supported the patentability of computer-implemented inventions. Because the "any hardware" approach radically limited the scope of the Art. 52(2) exclusions by shifting the emphasis to the actual technical contribution of the invention, this approach removed almost all subject matter related to computer programs (and methods of doing business) from the excluded categories. The "any hardware" approach has sparked debate on whether putting meaningless physical limits on a claim (i.e., suggesting that software is technical simply because it runs using hardware) converts non-technical or abstract ideas into technical or concrete ones.

Some remarks should also be made regarding the national decisions made in the courts of the EPO member states. The European national patent legislation is generally in line with international treaties such as the European Patent Convention (EPC). However, European patent law is still enforced nationally. Accordingly, discrepancies may arise from the different interpretations by the national courts. These existing divergences in the field of CII were highlighted in 2006 during the Aerotel/Macrossan case in Britain[67]. The case involved the patentability of computer programs and business methods. Overall, this highly debated ruling has attracted worldwide attention to the Art. 52(2) exclusions, particularly with respect to CII and business methods. The judgement pointed out the inconsistency of the approaches followed by the EPO Boards of Appeal regarding the patentability of computer programs and to the lack of well-defined rules in Europe regarding patentable subject matter, especially with respect to software.

The fierce debates initiated by Aerotel/Macrossan in Europe led to a referral to the EPO Enlarged Board of Appeal in October 2008 that sought clarity regarding the applicable rules for CII patents[68]. However, the Enlarged Board's decision that was delivered on May 2010 failed to provide clear guidelines regarding patentable subject matters. In fact, the Enlarged Board found all of the questions posed to be wholly inadmissible on the grounds that there was no divergence from the EPO case law. Thus, the board concluded that the legal requirements for a referral were not met.[69]

Europe now seems closer than ever to developing a unitary patent system. The discussion has continued for more than five decades, and several proposals to create a unitary European patent litigation system have been made. The fragmentation problems deriving from not having a unitary patent enforcement system in Europe allegedly distort the internal market and encourage uncertainty. Overall, this situation makes it challenging for companies, users, courts and legislators to operate in the system. All of these possible negative effects seem to be accentuated in technology-related fields such as the software sector, where there are no generally accepted, clear rules regarding patentability.

Among the most recent initiatives are two proposed patent reforms that are currently being discussed in the European Union: the unified European patent system and the Unified Patent Court. Under the former, European patents granted by the EPO under

---

[67] Court of Appeal (England and Wales), *Aerotel Ltd vs. Telco* [2006] EWCA Civ 1371.

[68] See G 3/ 08 "Referral under Article 112(1)(b) EPC" (23 October 2008). It should be noted that in the decision on Aerotel/Macrossan, Justice Jacob recommended a referral to the Enlarged Board based on the inconsistent approaches followed by the EPO Boards of Appeal with regard to CII patentability. However, on this occasion, the EPO President (at that time, Alain Pompidou) refused.

[69] See EPO Case Law of the Boards of Appeal, Headnote of opinion G 3/08 at: http://www.epo.org/law-practice/legal-texts/html/caselaw/2010/e/clr_headwords.htm.

the EPC could also apply in the participating Member States. These European patents would co-exist with national patents and (classic) European patents. Furthermore, the EPO would be given new tasks, including the solicitation, registration, publication, collection and distribution of renewal fees.[70] After the Advocate General issued an initial opinion in 2010 indicating that the proposal suggesting a combined court system for EPC and EU patents was incompatible with the EU Treaties, the Court of Justice concluded that:

> "The envisaged agreement creating a unified patent litigation system (currently called European and Community Patents Court) is not compatible with the provisions of the EU Treaty and the FEU Treaty"[71].

The EU Presidency presented a revised draft agreement regarding a Unified Patent Court on 19 October 2011.[72] According to this draft agreement, the Unified Patent Court would comprise the following: a patent court common to the participating EU Member States that addresses the disputes related to the (classical) European patents and the new universal European patents with unitary effect; a Court of First Instance, which is composed of a central division as well as several local and regional divisions set up in the participating Member States; and a centralised Court of Appeals. The Unified Patent Court would have exclusive jurisdiction over the (classical) European patents and the universal European patents. The Court would defer to EU law and would request preliminary rulings from the CJEU in accordance with Article 267 TFEU. The Member States would be held accountable for any infringements on EU law by the Unified Patent Court.

2.2.2.3. Limits of Patent Protection

Patents are an efficient instrument for protecting innovation by encouraging technological progress. However, patents work differently depending on the industry to which they are applied. In some technological fields (e.g., the pharmaceutical industry), it would be impossible to stimulate innovation without patent protection.[73] When new technologies and thus new types of inventions arise, however, moulding the system to meet the features of the new developments may be a challenge. In these cases, it is particularly important to define the precise consequences of allowing patents for such inventions[74]. No proprietary system is morally justifiable if its operating costs are higher than the benefits for the whole of society.

Indeed, it has been argued that software innovation has brought the patent system to its knees and that the latter has completely failed in this area.[75] As will be explained in section 4.1.2., the problem is reflected at various levels of the patent process (i.e., from the initial filing of the patent applications to the enforcement phase). Notably, the

---

[70] See The Council of the European Union, "Proposal for a Regulation of the Council and the European Parliament implementing enhanced cooperation in the area of the creation of unitary patent protection", 11328/11 (23 June, 2011).
[71] Avis 1/09 of the European Court of Justice, "Creation of a United Patent Litigation System" (8 March, 2011).
[72] See The Council of the European Union, "Draft Agreement of a Unified Patent Court and Draft Statute – Revised Presidency Text" (19 October, 2011).
[73] Alison J, Dunn A, and Mann R, "Patents and Business Models for Software Firms" (2006), *The University of Texas School of Law*, Law and Economics Research Paper No. 77.
[74] Mann R, "Do Patents Facilitate Financing in the Software Industry?" (2005), 83 *Texas Law Review* 4.
[75] See note 4 above at 3-6.

problems with patents and software have been reported more vigorously in the United States, where few restrictions on the patenting of software (and business methods) have been imposed since the mid-1990s. In contrast, the European limitations dictated by the "as such" exclusions have been suggested to play a positive role in reducing these concerns.

2.2.3. FOSS: Viable Alternative or Balancing Force?

The concept of "free" software, conceived of as the free exchange of source code for computer programs, dates back to the origins of software development. As previously mentioned, the first computer programs were developed by computer scientists who exercised openness and free sharing. However, what is now named "free *libre* open source software" (FLOSS/FOSS/OSS)[76] refers to the free software movement and the GNU Project[77], which was founded in 1983 by Richard Stallman, who was a computer scientist at MIT laboratories at that time. To support the movement, Stallman founded the Free Software Foundation (FSF) in 1985.

FOSS originated from the perceived need to "fight" against proprietary software, especially patents. The primary goal of FOSS is to ensure the four "essential" freedoms for computer developers and users and is summarised as follows:

- The freedom to run the program for any purpose (freedom 0);
- The freedom to study how the program works and to change it such that it computes as you wish (freedom 1);
- The freedom to redistribute copies such that you can help your neighbour (freedom 2). Access to the source code is a precondition for this freedom;
- The freedom to distribute copies of your modified versions to other people (freedom 3). By doing so, you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this freedom.[78]

Software licensed under all of the abovementioned conditions is considered "free software". Free software is not the same as the absence of commercialisation. Since the creation of FOSS, several business models have been developed to take advantage of the resources available through FOSS by adding proprietary features that can generate revenue. Even if some FOSS projects are generated almost completely by volunteers, the ties between open source communities and proprietary firms have increased rapidly.[79] Many independent software vendors, hardware vendors, and value-added resellers use open source software in their proprietary, for-profit software products and services. This model is neither purely open source nor purely proprietary; instead, it incorporates both FOSS and IP. We refer to this model as the *hybrid* protection model.

---

[76] Free and Open Source Software (FOSS), Free, *Libre* Open Source Software (FLOSS), Open Source Software (OSS) are all slightly different alternatives used to describe software that can be used, modified and redistributed with few or no restrictions. For the purpose of this dissertation, these terms will be used interchangeably.

[77] See http://www.gnu.org/.

[78] "Free Software Definition", GNU Operating System at: http://www.gnu.org/philosophy/free-sw.html.

[79] Mann R, "Commercializing Open Source Software: Do Property Rights Still Matter?" (2006), 20 *Harvard Journal of Law & Technology* 1.

2.2.3.1. FOSS Licensing Model

Today, the term "open source" refers to various types of licenses. To be recognised as open source, licenses need to be certified by the Open Source Initiative, a non-profit organisation dedicated to promoting open source software.[80] Open source licenses differ from proprietary licenses in that the former does not restrict the rights of the licensees to specific uses, geographical areas, or products. Moreover, open source licenses do not contain marketing arrangements specifying the conditions for the payment of royalties to the licensor. Instead, FOSS licenses grant licensees the freedoms to use, copy, modify, distribute and redistribute the licensed software for any purpose and to distribute said software without paying royalties to the licensor.[81]

Each FOSS license is based on copyright law. FOSS licenses determine the relationship between the copyright holder and the users. Generally speaking, retaining the copyright for the original code renders FOSS licenses enforceable: the original developer retains the copyright for the original program, whereas subsequent developers retain the copyright for their improvements. If one does not comply with the license, then the license terminates, making it impossible to copy, modify, distribute, or redistribute the code without violating the owner's copyright.[82] These types of licenses are often referred to as "copyleft" licenses to describe the practice of using copyright law to offer the four fundamental freedoms.[83]

Nonetheless, FOSS licenses differ in several respects. Under commercialisation, the most important limitations on open source licenses are most likely those imposed on the use of FOSS-licensed code in other (proprietary) software. This feature differentiates "reciprocal" or "copyleft" licenses from "academic" licenses. "Copy-left" licenses are derived from the GNU General Public License (GPL)[84], which was originally written in 1989 by Richard Stallman for programs released as part of the GNU Project, whereas the "academic families" originated from the Berkeley Software Distribution (BSD) license[85], which was generated by Bill Joy at the University of California at Berkeley. The GPL and the BSD are the most frequently used OSS licenses.

The GPL and "copyleft" licenses are the most restrictive FOSS licenses with regard to the rights of licensees to license derivative inventions. The concept of reciprocal obligation included in the "copyleft" clause is essential. The GPL's restrictions apply not only to the original GPLed code but also to any "modified work" that includes GPL code, unless the modified code (or "identifiable sections" of it) "can be reasonably considered independent and separate works in themselves"[86]. No court case has interpreted the application of the "copyleft" clause in practice. The jurisprudence has merely discussed issues related to the failure of distributors to make certain GPLed code available. When defendants have been found to distribute code in breach of the

[80] See http://www.opensource.org/.

[81] See Weber S, *The Success of Open Source* (First Printing eds., 2004), Harvard University Press, Chapter 2. See also Välimäki M, *The Rise of Open Source Licensing. A Challenge to the Use of Intellectual Property in the Software Industry* (2005), Turre Publishing, Chapter 2.

[82] See GNU General Public License, version 3, at: http://www.gnu.org/copyleft/gpl.html

[83] *Ibid.*

[84] *Ibid.*

[85] See http://www.opensource.org/licenses/bsd-license.php.

[86] See GNU Lesser General Public License, version 3, "Terms and Conditions for Copying, Distribution, and Modification", 2b): "You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License".

GPL license terms, they have been asked to either start distributing it in compliance with the GPL (and to pay some license fees) or to stop distributing their own proprietary code under the GPL.[87] Additionally, many FOSS rights holders initially attempt to make reasonable arrangements with the alleged infringers. Usually, the rights holder will ask the infringing company to comply with the license before initiating legal proceedings.[88] However, the wording of the GPL imposes constraints on the developer's ability to incorporate GPL code into fully proprietary code.[89]

Generally, "academic" licenses provide more opportunities for commercial exploitation. In fact, the governing principle of these licenses is that everyone should be free to use work prepared solely for academic purposes without restriction.[90] Some academic licenses, such as the Mozilla Public License (MPL)[91] or the Apache License[92], state this concept explicitly[93], whereas others, such as the BSD, do not.

Another important difference between the different FOSS licenses is related to intellectual property rights, particularly patent rights. Certain FOSS licenses, such as the GPL and the BSD, rely on the traditional idea that any party contributing to a FOSS project has an implied right to the ordinary use of the licensed software. Other licenses, such as the Apache license, address the issue more directly by specifically requiring IP licenses from all contributors to the FOSS project. The latter type of licenses carefully limits the rights granted to certain aspects of the FOSS-licensed source code.[94] For instance, IBM's Common Public License explicitly excludes licenses for any patent not issued at the time of the contribution: "This patent license shall apply to the combination to the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition to the Contribution causes such combination to be covered by the Licensed Patents"[95]. However, unlike licenses for proprietary software, FOSS licenses do not usually provide for indemnification or make guarantees associated with IPR infringement, although they impose the risks of infringement on the licensees.

---

[87] For examples of court decisions reiterating these principles, see *Palnetary Motion, Inc. vs. Techsplosion, Inc.* (2001), United States Court of Appeal for the Eleventh Circuit, 261 F.3d 1188; *Progress Software Corp. vs. MySQL AB* (2001), Civil Action No.: 01-11031 PBS; *Computer Associates, Inc. vs. Quest Software, Inc., et. al.* (2004), No.: 02 C 4721; *In Re Welte vs. Sitecome Germany*, District Court of Münhen I , No.: 21 0 6123/04 (May 19, 2004); *In Re Welte vs. D-Link Germany*, District Court of Frankfurt am Main, No.: 2-6 0 224/06 (September 22, 2006); *Welte vs. Skype Technologies SA*, Higher Regional Court of Munich (May 08, 2008).

[88] For instance, in *FSF vs. Cisco* (USA, 2008), before initiating the legal proceedings, the OSS right holder asked Cisco for over two years to start distributing in compliance. The case was settled later in 2009.

[89] See Lerner J, and Tirole J, "The Scope of Open Source Licensing" (2005), 21 *J. L. ECON. & ORG.* 20.

[90] Eisenberg R, "Academic Freedom and Academic Values in Sponsored Research" (1988), 66 *Tex. L. Rev.* 1363.

[91] See Mozilla Public License, version 2.0, at http://www.mozilla.org/MPL/2.0/.

[92] See Apache License, version 2.0, § 4: "You might reproduce and distribute copies of the Work or Derivative Works therof in any medium, with or without modifications, and in Source or Object form [...]" (at: http://www.apache.org/licenses/LICENSE-2.0.html).

[93] See MPL § 3.3: "You might create and distribute a Larger Work under terms of your choice, provided that you also comply with the requirements of this License for the Covered Software [...]".

[94] For instance, the MPL limits the patent grant of the initial developer (i.e., Netscape) to cover only the patents that are necessary to the use of the Original Code.

[95] See Common Public License, version 1.0, § 2, at: http://www.eclipse.org/legal/cpl-v10.html.

2.2.3.2. Integrating the Protection Models

The pluralistic nature of software has led to the use of multiple protection mechanisms. As previously mentioned, under current rules, IP instruments such as copyright, patents, trademarks, licensing agreements, and technical protection measures are commonly used with software. Today, even open source software is used by proprietary software companies. However, combining several protection mechanisms may sometimes engender conflict. This problem can occur if IP mechanisms (especially copyright and patents) and FOSS are integrated into the same software package. It can be argued that the value of intellectual property increases through free sharing. However, legal issues can arise because of the different principles upon which the two types of mechanisms are based. As will be subsequently discussed in section 4.1.2.1., there is still confusion regarding the interpretation of many FOSS licenses terms, which may increase (or reduce) the likelihood that companies will face legal risks.

3 SUMMARY OF THE ESSAYS

The following section summarises the overall objectives and the main contributions of the individual essays within the thesis. Subsequently, section 4 discusses the overall contribution of the study, the practical implications of the research results, and their limitations.

3.1. Essay I: Scope of IP Protection for the Functional Elements of Software

Abstract

It is well-known that the pluralistic nature of computer program code has made it challenging to keep up the distinction between copyrightable and patentable aspects of software. The dual nature of software, as both literary and performing (practical) functions at the same time, has been particularly problematic for intellectual property laws.

This article argues that both European copyright and patent laws urgently need some refinement in order for them to interact with each other in a way that would promote rather than impede innovation in such a fundamental field of technology as computer software. The more pressing questions concern the scope to be accorded to copyright and patents, particularly with respect to the protection of software's most valuable assets, namely its functional elements. It is evident that when it comes to functional aspects, patents are a much better protection mechanism than copyright. However, in order to maintain the distinction between copyrightable and patentable objects in software two rules should be considered. Firstly, on the copyright front, this article advocates the adoption of explicit statutory limitations against non-literal infringement to keep the scope of software copyright protection within the sphere of literary works of art. Secondly, the paper suggests that keeping software within the umbrella of patent law is essential in order to impede copyright from expanding to protect the functional elements of software in a "patent-like" manner.

3.1.1. Overall Objectives and Main Contributions

This article focuses on the intellectual property protection of the functional aspects of software. More precisely, this article focuses on the scope of copyright and patent rights. Specific attention is paid to the problems that the IP system faces because of the dual literary-functional nature of computer programs. The paper discusses two of the most controversial issues related to software copyright: the failure of copyright law to protect the most valuable parts of the code (i.e., its functional elements) and the difficulty of drawing boundaries between the functional and expressive elements of a computer program. The paper argues that the solution lies not only in copyright law but also in the interaction between the copyright and patent regimes for computer programs.

Copyright and patents are the intellectual property mechanisms that are most commonly used to provide legal protection to computer programs. However, the scope of copyright differs vastly from that of patents. The scope of copyright is defined by the rather vague "idea-expression dichotomy", under which copyright protects the

expression of ideas and should not extend to the ideas *per se*. Conversely, patents protect the implementation of ideas and apply to functional, non-literary objects. The scope of patent protection is defined by the claims.

From a theoretical perspective, if a single creation includes both protectable ideas and protectable expressions, each regime within IP law should apply to different, mutually exclusive aspects of the creation. However, although distinguishing between the literary and non-literary elements is simple for those creations in which the expression of ideas and the ideas themselves can be fully separated, problems arise if the expression is an integral part of the idea itself such that the two are inseparable. This case often arises in software: program code is pluralistic in nature and features inseparable literary and functional elements.

The paper identifies two specific sets of problems inherent to software copyright:

- The risks of under-protection: this issue arises if the copyright extends to only the source or object code and does not protect the way that the program works. The most valuable parts of any software program are its functional, utilitarian elements. Because copyright does not protect companies from having their technical information shared, competitors can determine how their products work and reproduce them without infringement.

- Risks of over-protection: this issue arises because of the difficulty of drawing boundaries between the literary and functional aspects of software. This problem has led copyright law to extend occasionally to functions and thereby protect software in a "patent-like" manner. Extending copyright law to protect functional works of technology not only contravenes the traditional principles of copyright and explicit statutory provisions but might also be highly detrimental to software innovation because of the broad scope and long duration of copyright protection.

The abovementioned risks are particularly problematic in Europe. The traditional reluctance to afford patent protection to computer programs under Article 52(2)(3) of the European Patent Convention (EPC) has created a system that tends to favour a broad scope for software copyright. The Software Copyright Directive leaves open the possibility of broadly interpreting the scope of software copyright at the national level. However, the national rulings have been scarce and inconsistent thus far. As a result, the European courts are still far from developing a uniform position on these issues.

To solve these problems and enhance the development of and progress in the software sector, the European copyright and patent laws must change. The most pressing question is what scope should be afforded to copyright and patent protection. This paper suggests that two specific rules be considered for the European framework.

First, the article advocates the adoption of explicit statutory limitations on non-literal infringement (i.e. direct copying) that will keep software copyright protection limited to literary works of art. Whether the non-literal copying of code needs to remain under the umbrella of copyright law is truly unclear. Additionally, confining copyright to literal infringement would ensure that copyright rules apply to only the software's expressive elements and not to its functions.

Second, the paper argues that maintaining patent protection for software is important because patents provide an adequate level of protection for the most valuable aspects of

software (i.e., its functional elements) and because patents promote a narrow interpretation of the scope of protection afforded by software copyright. The paper suggests that this approach may be the only workable method of maintaining the distinction between copyrightable and patentable objects in software. The debate over software as patentable subject matter in Europe is irrelevant. The software sector would benefit much more from concentrating its valuable efforts on improving the current imperfect patent system.

3.2. Essay II: Software Patents in Europe. The Technical Requirement Dilemma

Abstract

The ongoing debate over the patentability of computer-related inventions clearly demonstrates that this is anything but a dead issue in Europe. At stake, in particular, is the exclusion of computer programs "as such" from the patentability field, and its traditional interpretation based on the requirement of "technicality".

This article pursues the subject, first investigating what type of inventions involving software have been considered technical by the EPO, on the one hand, and by some national courts, on the other. On this respect, the practices followed by the UK, Germany, and France, European countries where most of software patent litigation takes place, are considered. The paper analyses the consequences deriving from the European obstinacy in trying to stretch and adapt old rules to completely new situations. The study concludes that the traditional European system, where only "technical" inventions receive patent status, has become obsolete under current rules. A radical change is, therefore, an actual priority.

3.2.1. Overall Objectives and Main Contributions

The essay addresses the issues related to treating computer programs as patentable subject matter under the European rules. Specifically, the paper investigates the reasons for and consequences of the exclusion of computer programs "as such" from patent law under Article 52(2)(3) of the European Patent Convention and the related problems resulting from the application of the "technical" criterion to software-related inventions. The requirement of technicality is embedded in European patent law, and its application has proven to be anything but trivial in the field of computer-implemented inventions (CII).

The exclusion of computer programs "as such" under Article 52 EPC does not indicate that patents are rarely applied for or granted in the software field. The interpretation of how a computer program "as such" should be characterised has generated in-depth discussions for the past three decades. Although both the EPO Boards of Appeal and the national courts of the EPO member states have issued a series of relevant decisions, the precise definition of "technical character and/or contribution and/or effect" remains unclear. Additionally, the criterion of technicality has often led to arbitrary and contradictory decisions.

The article provides a detailed analysis of the evolution of the jurisprudence related to the patentability of computer programs at the EPO and in three national jurisdictions: the UK, Germany, and France. This analysis reveals the various and inconsistent

approaches embraced by the EPO over the years and the significant differences that persist between EPO case law and national European jurisprudence. A general lack of legal consensus exists regarding CII patentability within a potentially fragmented European system, and one could argue that this disharmony may distort the European market and impede economic growth in the software field.

On this basis, the paper formulates a more workable approach that would enhance the clarity of Art. 52's dictate in the field of computer programs. Specifically, the paper suggests that the traditional concept of the "inventive step" be reformulated and recommends limiting the scope of protection for software patents. First, to facilitate the assessment of "technical invention" in the software field, the paper advocates the "any hardware" approach embraced by the EPO in the 2000s. According to this approach, software-related inventions involving the use of some type of hardware (e.g., a general computer) are considered technical, and the technicality of the invention is determined based on its inventiveness. Under the "any hardware approach", the seemingly absolutist "as such" exclusion of computer programs loses its meaning, but this method appears at first sight an easier way of interpreting European patent law without violating the EPC principles. Nevertheless, the paper notes that the problem-and-solution-approach used to assess inventiveness at the EPO on a predominantly technical basis does not solve the "technical/non-technical" dilemma. To be considered inventive under European patent law, an invention must provide "a technical solution to a technical problem" (under the problem-and-solution approach). Indeed, the adapted version of the problem-and-solution approach developed by the EPO in the COMVIK case soon after the launching of the "any hardware" approach poses special consideration for applications comprising mixed types of features (technical and non-technical features). According to COMVIK, in fact, it is legitimate to have a mix type of technical and non-technical features in the claim. In order to address the question of inventiveness, however, not only the technical and non-technical features must be clearly separated at first, but also inventiveness can be based only on the technical features, while features that contribute only to the solution of a non-technical problem (e.g. business and administration) cannot support the presence of an inventive step. Indeed, the "technical/non-technical" dilemma (intrinsically related to the definition of "technology") remains inculcated into the system. This way, instead of solving the long lasting problems with CII and patent rules, the "any hardware" and COMVIK approaches simply transpose the dilemma from the requirement of invention to the requirement of inventiveness. To resolve this problem, the article suggests assessing inventiveness after considering all of the invention's technical and non-technical features to promote more coherent and transparent practices. Such a system would eliminate the impossible task of separating the technical features from the non-technical ones (this way eliminating the difficulty enshrined with defining "technology") and allow examiners to concentrate on deciding whether the invention is actually inventive with respect to the current state of the art.

In the software field, where innovation requires small steps rather than large breakthroughs, the inventiveness bar should be set quite high. Accordingly, the patent scope should be limited, and patents should not apply to several product generations. This proposed harmonised system focuses mainly on inventiveness by using a relatively narrow patent scope and would provide a reliable basis for software patent protection. The system would also eliminate the contradictions that currently arise because of the application of the technical criterion in Europe. Moreover, this system would increase transparency and patent quality to the overall benefit of companies, consumers, and the European Community as a whole.

The essay advocates a narrow interpretation of the patent scope under the framework of the European Patent Litigation Agreement (EPLA). This proposed patent law agreement aimed to create an additional EPC protocol that would require the signatory member states to commit to using an integrated judicial system, uniform rules of procedure and a common court of appeals. The EPLA project seems to have been abandoned under the current rules. Nevertheless, as mentioned in the first section of this introduction, other similar suggestions, such as the proposal for the Unified Patent Court, have replaced the EPLA option, even though the focus has shifted from the EPO to the framework of the European Union. Accordingly, the proposal in this essay could be considered as part of the framework of the Unitary Patent Court.

3.3. Essay III: The Software Patent Thicket: a Matter of Disclosure

Abstract

The high complexity of software products, as well as the increased number of intellectual property rights in the field, has created a dense thicket of overlapping patent claims that companies must navigate in order to operate in the sector. The lack of relevant prior art and the abstract nature of the software patent claims are the major causes of overlapping patents in the field. However, efforts have thus far been concentrated merely in improving the prior art repositories. The abstract nature of the patent claims and the disclosure concerns deriving from that have, however, not yet received sufficient attention. This article pursues this subject, first by investigating the reasons for, and consequences of, overlapping IP rights in software-related patents. This analysis suggests that overlapping problems and, thus, the software patent thicket, cannot be effectively reduced unless issues related to abstraction and disclosure are addressed. On the basis of this, a more detailed description of the program – including flowcharts, pseudocodes and, when necessary, parts of the source code – might be an essential requirement to be added to the description of the invention in natural language.

3.3.1. Overall Objectives and Main Contributions

The article aims to tackle one of the major problems with software-related patents: "patent thickets". The article addresses issues related to software patents and patent thickets in general rather than patent thickets in the software industry *per se* because the majority of software-related patents are obtained by firms operating outside of the software industry. Thus, the software patent thicket may not necessarily be reflected in the software industry *per se*. Indeed, software-related patents create patent thickets in other sectors, such as semiconductors and computer hardware, which own most of their software patents.

Usually, software inventions are highly complex and include several components. These components are often protected by multiple intellectual property rights. Consequently, numerous licenses might be required to produce even a single commercial product.[96] Furthermore, together with the concerns surrounding the "quality" of many of the patents issued in the field, the vast proliferation of software-related patents in a relatively short period has allegedly created dense overlaps with regard to patent

[96] See note 34 above.

claims. The economist Carl Shapiro named this situation "patent thickets" and defined them as follows:

> "A dense web of overlapping intellectual property rights that a company must hack its way through in order to actually commercialize new technology."[97]

Patent thickets are caused by overlapping patent rights and by the large volume of patents being issued. This article finds that the main reasons for the overlap in software patents are the lack of relevant prior art in the field and the abstractness of software patent claims. The great volume of software-related patents is determined based on available statistics and studies.[98]

The article suggests that although several studies have been conducted and initiatives have been put into place to improve the prior art repositories and allow for better searching of computer programs, the abstractness of software patent claims has not been sufficiently considered. Additionally, the article demonstrates that abstraction is one of the major causes of the growing software patent thicket. As a result, abstraction represents a fundamental issue to be addressed. In particular, abstraction might lead to three specific problems for software technology:

- Abstractness leads to inventors claiming technologies that are often not yet known to the inventors at the time of the application. As a result, patents may be issued broadly, and inventors may be rewarded for things that they did not invent.

- Abstraction makes it hard for the examiners to judge whether enough information has been provided to support the applications properly.

- Finally, this configuration makes it difficult for patent officers to draw the boundary between allegedly new inventions and prior art. The abstract nature of software patent claims leads to the issuing of patents that do not possess clear boundaries and thus gives rise to high risks of infringement and opportunistic litigation.

The article pursues the thesis that overlapping problems (and thus the software patent thicket) cannot be effectively reduced unless the issue of abstractness is properly addressed. Indeed, the paper suggests that the more abstract the claims are, the more difficult it is to evaluate the concrete applicability of the inventions and the more interpretation is required to assess the question of patentability. Accordingly, extensive disclosure would seem to be necessary in software patent applications to facilitate the process. The article identifies a major failure of the patent system: the relatively low level of disclosure needed to file a software-related patent (both in the U.S. and in Europe). Poor disclosure has played a fundamental role in intensifying patent overlap and patent thickets in the field.

---

[97] Shapiro C, "Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard–Setting" (2001), 1 *Innovation Policy and the Economy*, at 118-150.

[98] See, for instance, Mann R note 79 above. See also Bessen J, and Hunt R, "An Empirical Look at Software Patents" (2007), 16 *Journal of Economics and Management Strategy* 1, at 157-189; Bergstra J, and Klint P, "About 'Trivial' Software Patents: The IsNot Case" (2007), 64 *Science of Computer Programming* 3, at 264-285; Merges R, "Intellectual Property Rights and Bargaining Breakdown: the Case of Blocking Patents" (1994), 62 *Tennessee Law Review* 1, at 75-106; Burk D and Lemley M, "Designing Optimal Software Patents", in Hahn R, *Intellectual Property Rights in Frontier Industries: Software and Biotechnology* (AEI Press 2005).

On this basis, the article suggests that the disclosure requirements for computer program patents must be enhanced to reduce the abstract nature of software patents claims, to better define the boundaries between patentable inventions and prior art, and to help narrow the scope of the patents granted. Furthermore, increased notice would reduce the number of applications filed and thus also indirectly smooth the thicket. Finally, enhanced disclosure might also be necessary for software-related patents to fulfil the purposes of patent law. Failing to meet a sufficient disclosure threshold might well be considered unlawful. To achieve this target, the paper suggests that a more detailed description of the program, including flowcharts, pseudocodes and, if necessary, parts of the source code, should be added to the description of the invention in natural language. The proposed reform should primarily originate from the patent offices. Accordingly, examiners should ask for more information about the meaning of the claims and should reject vague and abstract claims more aggressively.

Overall, the proposed solution would help patent officers better evaluate applications and would help judges address questions of infringement. Competitors and new inventors who need detailed information about the programs behind patented inventions to improve upon them would also clearly benefit from this practice.

3.4. Essay IV: Proprietary Software vs. FOSS: Challenges with *Hybrid* Protection Models

Abstract

This article analyses the reasons and consequences of the fact that open source software has become a portion of the technology used by proprietary companies. It focuses on problems arising from the use of what is designated here as the *hybrid* protection model by commercial companies. The term *hybrid* model refers to a situation where companies incorporate both open source and IP protected code into the final proprietary software they release to the market. The coexistence of both the proprietary and the open source software model is essential to promote innovation in the software field. Due to the different and allegedly conflicting principles under which they are based, however, the relationship within the two systems might not always be peaceful. By combining legal theory and empirical research, this paper provides a comprehensive analysis of the "core" legal challenges surrounding the implementation of the *hybrid* model in the context of commercial software, and sheds light on the coping mechanisms companies implement in order to navigate such risks.

3.4.1. Overall Objectives and Main Contributions

The article investigates the problems that may confront commercial companies that use the *hybrid* protection model. The *hybrid* model encourages companies to incorporate both open source and IP-protected code into the final proprietary software that they release to the market.

Both the advantages and the limitations of the proprietary and open source software protection mechanisms are initially investigated. The paper highlights the potential profits and the creation of consistent platforms upon which applications can be run as the major advantages of proprietary software. The disadvantages are the shortcomings of both copyright and patent protection with respect to software. The strengths of the open source mechanisms are identified as cost savings (from license, development, and

time costs), the generally high quality of the software in the widely used FOSS packages, the fact that the source code is made available to developers and users, and the availability of skilled developers, whereas the disadvantages include the generally scarce pecuniary rewards, the security concerns, and market fragmentation.

Overall, this analysis shows that the different and often conflicting aims of the proprietary and open source systems might hamper their peaceful coexistence once both models are embedded into a company's final proprietary software product. To shed light on the major legal challenges associated with *hybrid* models, this article conducts a practical analysis of company examples. This analysis is based on a literature review and shows some of the strategies that firms might use to mix open source and proprietary software. Specifically, three different groups of companies are identified:

- Traditional commercial (i.e., proprietary, "closed") companies that have modified their fully proprietary protection models to incorporate open source software. Here, the strategies implemented by International Business Machines (IBM) were analysed.

- Traditional FOSS firms that have started to use closed software. From this group, Red Hat was investigated.

- Purely *hybrid* companies that started with a mix of closed/open features. MySQL AB (currently owned by Oracle Corp.) is a representative example from this group.

The paper focuses on the legal challenges that can arise from the *hybrid* model and the coping mechanisms that could be implemented to navigate the risks. The paper formulates the potential problems that might arise from using *hybrid* models and provides empirical evidence of which legal risks are the most concrete and what coping mechanisms are used by companies in practice. Toward this end, an empirical case study was conducted. The case study was conducted with the intention of generating new knowledge rather than answering one (of a few) specific questions. The study was a multiple case study that used both documents and open-ended interviews as sources of evidence. The study considered six firms: one consultancy company and five software companies. The study sought to provide empirical evidence related to the use of *hybrid* models and the following concerns:

- Compliance and compatibility with license terms.

- The reciprocity of FOSS licenses.

- Ownership of rights-related concerns.

- Patent infringement and litigation risks.

- The dilution of the company's own patents.

- The lack of warranties and indemnifications on the FOSS side.

The empirical study confirmed the existence of all of the theoretically formulated legal risks, though some are considered more concrete than others. The study showed that no coping mechanism is fully foolproof *per se*. According to the findings, a well-

structured internal procedure and a solid model architecture and principles are the most effective way of reducing the legal challenges associated with *hybrid* models.

4 IMPLICATIONS, CONCLUSIONS AND FUTURE RESEARCH

This doctoral dissertation was written with four objectives in mind: 1) to improve understanding of the problems associated with applying intellectual property laws (copyright and patents) to computer programs; 2) to study how copyright and patents interact within each other's framework and within the overall IP framework when applied to computer programs (to introduce a new perspective to the overall discussion); 3) to propose practical, balanced and more workable solutions to the problems within the copyright and patent software frameworks in the European jurisdiction; and 4) to study whether the legal risks associated with the *hybrid* protection models are problematic in practice, to illuminate the causes of the issues at stake, and to assess which coping mechanisms could be used to alleviate these problems.

The findings related to the above research objectives were presented in detail in the previous section, which addressed the core objective and contributions of each individual study and will thus not be reiterated here. Instead, this session summarises the overall findings and discusses the overall contribution of the study.

This thesis has various theoretical and practical implications. The importance of computer programs around the world is undisputable. Currently, software can be found everywhere. However, software is still being developed. Thus, clear legal rules regarding software protection are more necessary than ever. The boundaries of intellectual property protection for computer programs remain murky in several respects. These problems have generated a global debate regarding the extent to which software should be afforded IP protection. Several perspectives have been presented, and there are many different opinions on the issue. The most extremist positions are the following: 1) because the current intellectual property systems of copyright and patents do not provide adequate protection to computer programs, software should be free for everybody to use; and 2) the only way to secure investments and growth in an important technological field such as software is to provide a very strong level of IP protection. Several more moderate proposals have also been suggested, including the creation of an *ad hoc* protection mechanism tailored specifically to computer programs and the modification of the existing IP rules such that they meet the needs of computer software. The major achievement of this thesis is that it effectively contributes to this debate by providing a number of balanced, reasonable answers to some of the major failures of the software intellectual property ecosystem. The proposed solutions were sought within the framework of the currently existing IP laws. In this way, this thesis provides relevant contributions to several bodies of IP law in the area of software protection, especially in the European framework.

4.1. The Software Intellectual Property Debate

As explained earlier in this introduction, software has been remarkably difficult to classify within a specific category of intellectual property. The issue has been discussed for more than thirty years, and the fact that recent developments have not yielded a solution shows the complexity of the debate and the problems involved.

This thesis has engaged some of the most controversial aspects of the debate surrounding the application of the most widely used IP protection mechanisms for

software: copyright and patents. The end result is a holistic study that actively and substantially contributes to the academic discussion of software IP protection. The findings are practical, readily available tools that, if implemented, might enhance clarity and legal security in this important field of technology. The results of the study could generally serve scholars, lawyers, software companies, and policymakers.

4.1.1. Copyright and Functions

Initially envisaged as an appropriate solution, copyright possesses several unavoidable shortcomings when applied to software. The most controversial aspect of copyright protection and computer programs is the "idea-expression" dichotomy. According to this well-affirmed principle, copyright protects the expression of ideas and should not extend to the ideas *per se*.[99] The application of this theory to computer programs has faced several impediments in many jurisdictions.

The primary problem is the pluralistic nature of software, which is simultaneously literary and functional. Computer programs are unique in that the software functions are fully independent from the literary lines of code. In other words, even though the source codes of two programs might look completely different, the codes can perform the exact same function and produce the same (or similar) instructions. Accordingly, drawing the line between the literary parts of programs and the mechanical ones has been extremely challenging.[100]

Over the years, the courts have struggled to provide a clear-cut answer to these questions.[101] This struggle is evidenced by the extensive American jurisprudence in particular. Notwithstanding the much less abundant case law, this same challenge has

---

[99] *Software Copyright Directive*, Art. 1.2 and Recital 14; 17 U.S.C. § 102(b) (2000).

[100] Samuelson P, *et al.*, "A Manifesto Concerning the Legal Protection of Computer Programs" (1994), 94 *Colum. L. Rev.* 8, at 2308-2431. See also Menell P, "Tailoring Legal Protection for Computer Software" (1986-1987), 39 *Stanford Law Review* 6, at 1329-1373.

[101] For U.S. relevant decisions, see: *Whelan Associates Inc vs. Jaslow Dental Laboratory Inc.* (1987), U.S. Court of Appeal for the Third Circuit, No.: 797 F.2d 1222, 230 U.S.P.Q. 481; *Synercome Technology, Inc. vs. University Computing Co*. (1978), U.S. District Court Northern District of Texas (Dallas Division), No.: 462 F. Supp. 1003, 199 U.S.P.Q. (BNA) 537; *Plains Cotton Coop. Assoc. vs. Goodpasture Computer Serv., Inc.* (1987), 807 F.2d 1256 (5th Cir. 1987) reh'g denied *en banc*, 813 F.2d 407 (5th Cir. 1987), cert. denied, 108 S. Ct. 80; *Computer Associates vs. Altai* (1992), 982 F 2d 693; *Chamberlain vs. Skylink* (2004), No.: 381 F.3d 1178, 2004 U.S. App. LEXIS 18513 (2004); *Lexmark International vs. Static Control Components* (2004), U.S. Court of Appeal for the Sixth Circuit, No.: 387 F 3d 522, 2004 U.S. App. LEXIS 22250. For some European case law, see, for instance, from the United Kingdom: *John Richardson Computers Ltd vs. Flanders and Chemtec Ltd* [1993] FSR 497; *Ibcos Computers Ltd vs. Barclays Mercantile Highland Finance Ltd* [1994] FSR 275; *Cantor Fitzgerald International vs. Tradition (UK) Ltd* [1999] Masons CLR 157; *Navitaire Inc vs. EasyJet Airlines Co* [2004] EWHC 1725 (Ch); and *Nova Production vs. Mazooma Games and Ors* [2007] EWHC Civ 219. From Germany, see: *Inkassoprogramm*, BGH (German Supreme Court) (09/05/1985), 87 GRUR 1041-1047, available in English at: (1986) 17 *IIC* 81, at 681; *Operating System- Betriebssystem*, BGH (04/10/1990), Case No.: I ZR 139/89, available in English at: (1991) 22 *IIC* 5, at 723; and *Show Format- Sendeformat*, BGH (26/06/2003), Case No.: I ZR 176/01, available in English at: (2004) 35 *IIC* 8, at 987. From France see: *Sybel Informatique vs. Oxalead and Prolepse*, Tribunale de Grande Instance de Paris, 3d ch., 2d sect., (12/11/1992), [1993], Expertises 109; Paris Court of Appeal, 4th ch. A., (23/11/1994), [1995] Expertises 36; *Agent Judiciaire du Tresor et Trace vs. Softmax*, Paris Court of Appeals, 4th ch., (31/05/1995), [1995] Expertises 311; *Marben GL vs. Cao Diffusion*, Meaux Commerciale Court, (17/12/1996), [1997] Expertises 122; and *Computer Assoc. Int'l vs. S.A.R.L.* Faster, No.: 519/95 , Tribunal de Commerce Bobigny, (20/01/1995).

clearly emerged on the European front as well. The main difficulty is that if only concepts and not the actual language are copied, the courts are put to the challenging test of having to distinguish the idea from its expression.

After attempting to resolve the issue of the scope of copyright protection for software for several years, the U.S. solved the debate in court, where it was affirmed that copyright should provide a "thin" level of protection for software and that copyright should not extend to program structure, sequences or organisation (SSO).[102] In particular, in *Computer Associates vs. Altai*, the court stated that copyright should not cover the areas in which ideas and expression merge and in which programmers have less design choice. Specifically, the court mentioned the following:

> "1) mechanical specifications of the computer on which a particular program is intended to run; 2) compatibility requirements of other programs with which a program is designed to operate in conjunction; 3) computer manufacturer's design standards; 4) demands of the industry being served; and 5) widely accepted programming practices within the computer industry".[103]

In other words, the court affirmed that interoperability is not covered by copyright law. However, these restrictions on copyright protection for software have been partially balanced by the introduction and acceptance of patent protection for the functional elements of programs.

In Europe, the issue was addressed in the Software Copyright Directive, which specifically states that "only the expression of a computer program is protected", whereas the "ideas and principles" are not.[104] The wording of the Directive is general and leaves room for national interpretation. However, the European case law regarding the scope of software copyright is not yet abundant. As a result, the few decisions addressing the issue have inconsistently afforded copyright protection to programs. Confusion remains with regard to which method should be adopted to determine infringement in cases of non-literal copying.

Regarding copyright law, this thesis provides new perspectives on the protection of the *functional elements* of computer programs, especially in the European context. As previously mentioned, recent court decisions both in Europe and in the U.S. have again highlighted the importance of the "idea-expression dichotomy" in software copyright and have thus confirmed that a consensus has not yet been reached. Indeed, the "idea-expression" dichotomy is an example of how the theoretical and practical applications of the law can differ greatly. Although it is easy to envisage that copyright should merely protect the expression of ideas and never expand to the ideas *per se*, software has made it difficult to apply this theory.

Several proposals that have suggested ways to resolve the issues related to software and copyright have been formulated. Some have attempted to formulate precise, highly complex doctrines that might help the courts determine the eligibility of computer programs for copyright. For example, the Abstraction-Filtration-Comparison test developed in Altai in the U.S. was one such attempt. Others have suggested that systems outside of the traditional copyright framework be used. For instance, the scope of software copyright is not an issue in the context of open source software because copyright is simply used as an instrument for enforcing FOSS licenses. Again, others

---

[102] See *Computer Associates vs. Altai*, note 101 above.

[103] *Ibid.* See also *Lotus Development Corp. vs. Borland International Inc.* (1995), U.S. Court of Appeal for the First Circuit, No.: 49 F.3d 807, 34 USPQ2d 1014.

[104] See Preamble of the Software Copyright Directive (1991).

have advocated the abolition of copyright protection and the creation of a completely new body of law to accommodate the special needs of software.[105]

This thesis sought a solution not only within the framework of the existing copyright law but also based on the patent rules. This thesis also proposes the adoption of explicit statutory limitations on the definition of non-literal infringement that will keep the scope of software copyright protection within the sphere of literary works of art. Furthermore, this thesis advocates keeping software within the umbrella of patent law to prevent copyright from expanding to protect the functional elements of software. In this way, the thesis promotes a simple model that would make it possible to apply the theory to practice.

A recent opinion of the European Court of Justice (EUCJ) on the *SAS Institute Inc. v. World Programming Ltd.* case supports the first part of the solution proposed in this thesis by indicating that the scope of software copyright with regard to functions should be interpreted narrowly. By following the reasoning previously delivered in the opinion of the Advocate General the EUCJ stated that:

> "The Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs must be interpreted as meaning that the functionalities of a computer program and the programming language are not eligible, as such, for copyright protection. It will be for the national court to examine whether, in reproducing these functionalities in its computer program, the author of the program has reproduced a substantial part of the elements of the first program which are the expression of the author's own intellectual creation."[106]

This argument is a well-reasoned and highly justified attempt to narrow the scope of software copyright protection in Europe. Furthermore, it is certainly indicative of the importance of the issues at stake that well-known scholars, such as Professor Samuelson, Vinje and Cornish, had supported the A-G opinion and felt that competition and innovation in the EU software industry would be seriously undermined if the scope of protection would be extended to the functions of software.[107]

Furthermore, the second part of the solution proposed in this thesis (i.e., maintaining patent protection for software to keep the scope of copyright narrow) is consistent with recent studies. For instance, the empirical study conducted by Lerner and Zhu[108] found that the firms affected by the diminution of copyright protection disproportionately accelerated their patenting behaviour in subsequent years. The reverse scenario could also be imagined: if patents are seldom granted or their scope of protection is drastically reduced, firms will start relying more on broad copyright protection. If patent and copyright protection are (to some extent) substitutable, in fact, the weakening of one form should increase reliance on the other. Together with the restrictions imposed on the patenting of software in Europe, the potentially broad scope of the European Software Copyright Directive confirms this theory.

---

[105] See note 100 above.

[106] See Case C-406/10 *SAS Institute Inc. vs. World Programming Ltd*, JUDGEMENT OF THE COURT (Grand Chamber), [2012] ECR 0.

[107] Samuelson P, Vinje T, and Cornish W, "Does Copyright Protection Under the EU Software Directive Extend to Computer Program Behaviour, Languages and Interfaces?" (February, 2012), *European Intellectual Property Review*. Available at SSRN at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1974890.

[108] Lerner J, and Zhu F, "What Is the Impact of Software Patent Shifts? Evidence from Lotus v. Borland" (October 1, 2005). Harvard NOM Working Paper No. 05-11. Available at SSRN at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=596144.

4.1.2. To Patent or Not to Patent?

> "Fighting patents one by one will never eliminate the danger of software patents any more than swatting mosquitoes will eliminate malaria".
>
> (*Richard Stallman*)
>
> "If people had understood how patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be a complete standstill today [...]. The solution to this is patent exchanges with large companies and patenting as much as we can".
>
> (*Bill Gates, 1991*)

The problems related to the "idea-expression" dichotomy in software copyright law and the associated risks of over- and under-protection have created the perceived need to extend patent protection (which, by definition, protects the functional aspects of inventions) to computer programs.

Valuable arguments have been formulated both in favour of and against the application of patent law to computer programs. These arguments are based on many different principles and consider different factors. Some refer to the intrinsic characteristics of both the software technology and the existing patent system. These characteristics include the special nature of both software technology *per se* and software innovation as well as issues related to the proper application of the patentability requirements, especially novelty and inventiveness. Indeed, one of the most controversial issues related to software and the structure of patent law is whether computer programs are (or should be) patentable. The interpretation of the patentable categories is at the centre of intense debates both in the United States and in Europe. However, this matter has been addressed from different angles from the two sides of the Atlantic, and divergent results have been reached. Correspondingly, the issue of software as patentable subject matter is also one of the central aspects investigated in this thesis. In particular, Essay II focuses on the issue from the European perspective.

As previously mentioned in section 2.2.1.3., the most controversial aspect of the patentability of software in Europe has traditionally been the interpretation of the "as such" exclusions (Article 52 EPC) and the requirement of "technicality". Several proposals have been made in the legislation, the relevant case law, and the literature. For instance, as previously noted, both the EPO and the national case law have employed different approaches (e.g., the technical contribution approach, the further technical effect approach, and the any hardware approach). However, as is often the case with vague, ill-defined legal concepts, the principle of software as patentable subject matter has been applied in an uneven manner in Europe. The Directive on the Patentability of Computer-Implemented Inventions[109] proposed by the European Commission attempted to resolve this issue. The Directive aimed to provide a more precise definition of the phrase "technical effect". The most controversial aspect of the Directive (and the one that led to its rejection) was Article 5, which was claimed to open the door to the patentability of computer programs "as such":

> "Member States shall ensure that a computer-implemented invention may be claimed as a product, that is as a programmed computer, a programmed computer network or other programmed apparatus, or as a process carried out by such a computer, computer network or apparatus through the execution of software".

---

[109] *Proposal for a Directive of the European Parliament and the Council on the Patentability of Computer-Implemented Inventions*, COM(2002) 92, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:52002PC0092:EN:NOT.

Suggestions regarding how to positively define "technical" are also found in the literature. For instance, Bakels suggests that although "technology" should be conceived of as a dynamic concept because it is evolving rapidly in ways that human beings cannot forecast, the law should be clear and precise to ensure legal certainty: "there ought to be a fixed "metarule" that determines the content of the technology criterion at a particular point in time".[110] To ensure legal certainty, Bakels suggests that the subject matter itself should be defined positively in a way that includes everything that is considered to be technical at a certain moment in time.

Other theories have suggested that scholars develop a robust and meaningful definition of the term "invention" under the EPC and an understanding of how individual subjects such as computer programs should be conceived of as inventions. For instance, Pila identifies the core question as "what does it mean to protect something as an invention" and argues that to interpret Article 52(2) and (3), one must develop "some independent idea of what constitutes an invention" and of "what makes a subject matter inherently suited (or otherwise) to patent protection".[111]

This doctoral dissertation does not support any positive definition of "invention", "subject matter", "technical", or "technology". Not only might such definitions be extremely difficult to formulate, as the previous attempts have proven, but any fixed definition may also risk becoming obsolete in a short period of time because technology evolves quite rapidly. This thesis suggests that the complexity of the "technical character" of CII stems from the interpretability of the patentability requirements of *invention* and *inventiveness*. The thesis also recognises the "special" nature of computer programs as inventions. However, the thesis identifies the major causes of the problem within the European patent system itself. As a result, the thesis aims to find solutions that will work within the European patent rules.

On the *invention* side, the thesis shows that defining the technical character of a computer-implemented invention is not important under the current approach. Indeed, it is easier to assess novelty and inventiveness immediately when determining whether an invention of this type is patentable. This book recognises this modern interpretation as a major achievement in the longstanding debate regarding the assessment of computer programs "as such" as patentable subject matter. However, the book contributes further to this interpretation by suggesting that *inventiveness* should also be redefined with regard to CII. To fully address the "technical" requirement dilemma, policymakers should assess *inventiveness* by considering all of the features of the invention (i.e., the technical and non-technical features). One could argue that the proposed model strongly resembles the one used in the United States and that it could consequently have negative consequences similar to those created by lenient interpretations of patentable subject matter (e.g., "anything under the sun"[112] is patentable). Among these potential problems, the most serious one is that software patents encourage the creation of "patent thickets": "a dense web of overlapping intellectual property rights that a company must hack its way through in order to actually commercialize new technology"[113].

The literature indicates that software patent thickets may create several problems at various stages of the patent process (i.e., from the initial filing of the patent

---

[110] Bakels R, "Software patentability: what are the right questions?" (2009), 31 *EIPR* 10, at 515.

[111] See Pila J, "On the European Requirement for Invention" (2010), *International Review of Intellectual Property and Competition Law*, Vol. 42.

[112] *Diamond vs. Chackrabarty* (1980), 447 S.Ct. 303.

[113] See note 97 above.

applications to the enforcement phase). Patent thickets may generally increase the cost of innovation in several ways. The dense overlapping of patent rights in the field obliges manufacturers to acquire many licenses from different owners to commercialise even a single useful product. This burden might impede further developments overall because the many different owners can block each other such that the protected software becomes underutilised.[114] As a consequence, incentive effects may decrease, slowing down technological progress. Moreover, numerous unclear patents and the resulting uncertainties may increase both infringement and the risk of litigation. Particularly relevant is the danger of even a single product or service infringing upon many patents. This situation might lead to "royalty stacking", as even a single product might bear multiple royalty burdens.[115] At the same time, patent floods and royalty stacking might create hold-up problems (i.e., an early patent holder might be able to limit subsequent innovators).[116] Anticipating the expected costs of such claims, a second innovator might choose to perform a sub-optimal level of R&D or even decide not to invest in the innovation at all.[117]

This thesis recognises the problem of patent thickets as an important factor. However, this research does not find any cause effect relationship between software patent thickets and expanding patentable subject matters. Moreover, these thickets do not arise because the patent system might consider both the technical and non-technical features of inventions when assessing inventiveness. Instead, this thesis identifies abstraction as the main cause of these thickets. To reduce abstraction, the patent system should make it possible to conduct better searches for prior art (to determine *novelty*) and enhance *disclosure*. The thesis argues that these efforts would drastically reduce the abstraction of software patents claims and thus reduce the patent thickets problem.

Commentators have suggested several other ways to decrease patent thickets. Of those strategies, most likely the most widely accepted approach has been the creation of patent pools or the establishment of licensing and cross-licensing agreements among the companies involved.[118] Once the anti-competitive effects of pools are carefully considered, patent pools and licensing schemes might be good ways of mitigating the negative effects of patent thickets in certain technological fields. However, none of these remedies eliminate the problem itself; instead, they provide a way of navigating the already existing thicket. In contrast, the model developed in this thesis tackles the problem of patent thickets in the software field at its origin. Thus, the thesis provides a solution that might effectively eradicate the causes of the phenomenon at their roots and thus eliminate (or radically reduce) software patent thickets.

---

[114] So-called "tragedy of the anticommons". See, for instance, Bailey R, "The Tragedy of the Anticommons: Do Patents Actually Impede Innovation?" (2007), *REASON MAGAZINE*.

[115] See note 34 above.

[116] Allison J, and Lemley M, "Extreme Value or Trolls on Top? The Characteristics of the Most Litigated Patents" (2009), 158 *University of Pennsylvania Law Review* 1.

[117] *Ibid.*

[118] See Shapiro C, note 97 above; Bessen J, "Patent Thickets: Strategic Patenting of Complex Technologies" (2003). Available at SSRN at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=327760; Cockburn I, and MacGarvie M, "Patents, Thickets and the Financing of Early-stage Firms: Evidence from the Software Industry" (2009), 18 *Journal of Economics & Management Strategy* 3, at 729-773.

4.1.2.1. Open Source vs. Patents?

The strongest reasons for opposing patents for computer programs originated with the Free and Open Source Software (FOSS) movement. As previously mentioned in section 2.2.3., the FOSS licensing model is based on copyright principles and originates from the perceived need to "fight" against proprietary software, particularly against patents. However, in recent years, it has become increasingly difficult to draw a clear boundary between open source and proprietary software because the two protection models are much more frequently used in the same software packages. Furthermore, several FOSS developers (e.g., Red Hat) have actively started to acquire patents for the software that they have developed. There may be several reasons for this trend. For example, one could argue that the coexistence of proprietary and open source software is essential to promoting innovation in the software field. However, the different and allegedly conflicting principles upon which the proprietary model (including copyright and patents) and the FOSS model are based might lead to conflicts and legal problems.

Thus far, the research has independently focused on either FOSS as a phenomenon or proprietary software. Few studies have investigated the relationship and interactions between the two. Prompted by the lack of adequate data and the perceived need to fill this deep gap, this thesis presents an in-depth study that investigates some of the major problems arising from the simultaneous coexistence of the FOSS and proprietary protection models for software.

The thesis contributes theoretical and practical knowledge not by providing specific solutions to the problems but rather by shedding light on the *most concrete shortcomings* that the open source model faces once it is combined with proprietary software in a commercial set-up. Although the study suggests some possible coping mechanisms that could be used to navigate the legal challenges associated with the model, it shows that none of these methods is fully foolproof *per se*. However, this finding is not (and was not originally meant to be) the most valuable contribution of the research. For instance, it is not realistic to claim that the findings of this study are applicable to all companies that implement *hybrid* types of protection models. Instead, the most valuable contribution of the study is the new theoretical and practical knowledge that it presents. This knowledge can help to strengthen the general understanding of the problems surrounding the use of *hybrid* models in the software field.

4.1.3. Some Remarks on the Sui Generis Type of Protection

This thesis aimed to develop the solutions to the research problems associated with the current body of IP laws. However, as previously mentioned, several scholars have suggested that the solution should be found in a *sui generis* type of protection that is specifically tailored to meet the special needs of computer programs.[119] The challenge of integrating software with the framework of traditional IP rules certainly makes the option of creating a new specific body of laws a highly tempting one. However, this thesis does not support a protection mechanism of this type for the following reasons:

- Even if a *sui generis* protection model were used, this mechanism would have to co-exist with the existing IP instruments. The clear, international acceptance of copyright protection shows the extent to which this mechanism

[119] See Samuelson P, *et al.*, note 100 above.

is entrenched in the system. Furthermore, copyright still serves some fundamental purposes. For instance, the fact that copyright offers a close substitute for businesses, such as SMEs, that cannot afford the highly expensive and time consuming process of obtaining a patent; that software which, though it might have been created through substantial effort, does not meet the patent system's high requirements would be left unprotected because of the unavailability of copyright; and that copyright is at the foundation of one of the most successful protection mechanisms in the software sector: the open source software model. On the other hand, notwithstanding the fact that the application of patent law to software is still a highly controversial topic, patent protection for computer-related inventions exists (at least to a certain extent) both in the U.S. and in Europe; it seems unlikely that this will change in the near future. Moreover, the utilitarian elements of software are surely better protected by patent law than copyright.

- Adding a *sui generis* type of protection to the existing IP mechanisms for software would further complicate the already unclear interrelationships among the multiple IP protection mechanisms.

- The creation of an industry-specific body of IP laws also raises concerns regarding whether such an approach should be considered for every technology that does not fall into the traditional IP framework. Software might well be exemplary of the failure of IP law to meet the needs of new technologies rather than being the only case. Biotechnology and nanotechnology are two of the many fields in which traditional IP law has proven to be insufficient. Special *ad hoc* bodies of laws have already been adopted for different types of technology in the past. The protection of databases in the EU and the protection of semiconductor chips in the USA are most likely the best known examples. However, fully modelling IP laws on an industry-specific basis would be difficult and would not necessarily be a good option. Moulding the IP law to technology in many different ways would create uncertainty and confusion. Furthermore, drawing the line among technologies would be extremely challenging given that a significant percentage of inventions are connected to more than one field and that new fields arise regularly.[120]

Therefore, notwithstanding the possible advantages of adopting a *sui generis* regime for software, the potential drawbacks suggest that this approach might not be a workable option. It would be more desirable to implement some of the suggestions by the advocates of the *sui generis* proposal when interpreting the existing systems of copyright and patent laws.

4.2. Avenues for Future Research

This study is a multi-perspective project that combines several research methods. The research questions have explored several branches of intellectual property law (patents, copyright, FOSS) in one specific field of technology: computer software. The issues at stake have been examined by using mixed research methods and by combining

[120] See Burk D, Lemley M, "Is Patent Law Technology Specific?" (2002), *Berkeley Technology Law Journal*, Vol. 17, at 1155.

traditional legal dogmatic, comparative law, historical, political, technological and economical approaches.

However, this study suffers from limitations and shortcomings. The limitations stem from the interdisciplinary nature of the thesis in itself and the exploration of various IP legal fields. Undoubtedly, the field of IP law that has been most discussed is patent law. The thesis discusses the controversial aspects of patent law in the software field: namely, the "technical" requirement in the context of "invention" and "inventiveness" under European patent law, the "novelty" requirement with respect to improvements to prior art searches for software (and business methods) patents, and the requirement of "disclosure" as a way to reduce the abstractness of software patent claims (as well as the overall problem of patent thickets). However, other important topics related to judicial patent law in the field of computer programs have not been considered. Though mentioned in Essays II and III, the questions of "novelty" and the "patent scope" of software-related patents have not been extensively discussed. In addition, the application of the doctrine of exhaustion to software-related patents has not been addressed in the context of this dissertation. These highly interesting and important questions constitute valuable avenues for further research.

Additionally, although the U.S. perspective has been considered throughout the project, the main scope of the thesis is the European context. This focus enhances the overall contribution of the thesis given the relatively scarce research on the topic in Europe and the greater availability of such research on America. However, notwithstanding the phenomenon of European and EU-level (and international) harmonisation, European patent law is still enforced at the national level. Furthermore, each European patent regime (and thus its interpretations of the laws) is associated with a particular legal history and other specific characteristics that reflect the innovation policy of each country. This thesis highlights these different perspectives by reporting the interpretations of patent law presented in three EU Member States: the UK, Germany, and France. However, although these three countries are highly representative, this analysis disregards the differences that might arise between these states and other European jurisdictions. Broadening the scope of the research conducted in this thesis to other countries would certainly be interesting.

Regarding the relationship between copyright and patent law, this thesis focuses on one central concern: the protection of the functional elements of software and the interpretation of the "idea-expression dichotomy". This issue is the most controversial unsolved problem within the European system. However, other interesting research questions could be considered. For instance, future scholars may examine the interpretation of the concept of "originality" with regard to computer programs. As previously mentioned, it is generally accepted that the quantitative or aesthetic merits of computer programs should not be relevant if copyright protection is granted. However, originality is not clearly defined in general, and there is no formal tool for evaluating originality. The definition of originality is complex in several fields but is even more complicated with regard to computer programs because it is so difficult to distinguish between functions and expressions. Interesting questions could be explored. For instance, when is a new, original computer program created? Is it created when the existing code is modified? Must the modified program work? Can one use someone else's pieces of code together with the code one has written oneself and still claim that the code is original?

Finally, the empirical study conducted in the fourth essay also has limitations. This study investigated the legal risks associated with the *hybrid* model of protection.

Because the specific field of research at the centre of the essay has not been extensively investigated in previous literature and because of the "grey" area surrounding the interpretation of the FOSS licenses terms, the case study relied upon a number of theoretical assumptions. One of the main challenges of researching areas with ill-defined legal principles is the lack of clear-cut interpretations of the norms at hand. Therefore, it is not realistic to claim that the findings of the study are applicable to all companies that implement *hybrid* types of protection. Indeed, much about software *hybrid* protection models still remains a mystery and warrants further research. Perhaps the more essential work to be performed in this area would involve replicating similar studies with different participants to test the replicability of the results. Another interesting avenue for research would be the exploration of the different aspects of the open-closed business model in the software environment. For instance, the economic or managerial aspects of the problem could be investigated. The research conducted in this paper constitutes a solid basis for further study that might investigate or address problems related to the *hybridisation* of other fields of technology. For example, the biotechnology sector is another field in which the closed-open innovation model is prominent but remains unexplored by researchers.

APPENDIX 1 LIST OF REFERENCES

BOOKS

Bainbridge D, *Intellectual Property* (6th eds., 2006), Longman

Bainbridge D, *Intellectual Property* (7th eds., 2008), Longman

Barzanó and Zanardo, *Brevetti e marchi* (2a edizione, 1999)

Beresford K, *Patenting Software Under the European Patent Convention* (2000), Sweet & Maxwell London

Bessen J, and Meurer M, *Patent Failure. How Judges, Bureaucrats, and Lawyers Put Innovators at Risk* (2008), Princeton University Press

Burk D, and Lemley M, *The Patent Crisis and How The Courts Can Solve It* (2009), The University of Chicago Press

Chisum D, *et al.*, *Principles of Patent Law. Cases and Materials* (2004), New York NY: Foundation Press

Cornish W, and Llewelyn, D, *Intellectual Property: Patents, Copyright, Trade Marks & Allied Rights* (5th eds., 2003), Sweet & Maxwell

Egg S, *Analysis of Dis/Agreement with Particular Reference to Law and Legal Theory* (First eds., 2003), Kluwer Academic Publishing

Friedrichsen G, Burchfield R, and Onions C, *The Oxford Dictionary of English Etymology* (1996), Oxford:OUP

Hoppen N, *Software Innovations and Patents. A Simulation Approach* (2005), Ibidem-Verlag, Stuttgart

Klami H, *Comparative Law and Legal Concepts, The Method and Limits of Comparative Law and its Connection with Legal Theory* (1981), Turku, Finland: Vammala

Landes W, and Posner R, *The Economic Structure of Intellectual Property Law* (2003), Harvard University Press

MacQueen H, Laurie G, Waelde C, and Brown A, *Contemporary Intellectual Property: Law and Policy* (1st eds., 2007), Oxford University Press

Mathely P, *Le nouveau droit franc¸ais des brevets d'invention* (1991), LINA

Merriam S, *Qualitative Research and Case Study Applications in Education* (1998), San Francisco: Jossey-Bass Publishers

Posner R, *Frontiers of Legal Theory* (2004), Harvard University Press

Sarvas R, *A software engineering view to the decisions of the European Patent Office concerning computer programs* (2001), Helsinki University of Technology

Soininen A, *Patents in the Information and Communications Technology Sector—Development Trends, Problem Areas and Pressures for Change* (2007), Lappeenranta University of Technology

Stake R, *The Art of Case Research* (1995), Sage Publications, Inc.

Välimäki M, *The Rise of Open Source Licensing. A Challenge to the Use of Intellectual Property in the Software Industry* (2005), Turre Publishing

Wandtke A-A, Bullinger W, *Praxiskommentar zum Urheberrecht* (2006), München: C. H. Beck

Weber S, *The Success of Open Source* (First Printing eds., 2004), Harvard University Press

Yin R, *Applications of Case Study Research* (Second eds., 2011), Sage Publications, Inc.

Yin R, *Case Study Research: Design and Methods*, Volume 5 of Applied social research methods series (Fourth eds., 2008), Sage Publications, Inc.

Zweigert K, and Kötz H, *An Introduction to Comparative Law* (1998), Oxford University Press

Örucu E, and Nelken D, *Comparative Law a Handbook* (2007), Hart Publishing

ARTICLES

Allison J, Dunn A, and Mann R, "Patents and Business Models for Software Firms" (2006), *The University of Texas School of Law*, Law and Economics Research paper No. 77

Allison J, and Lemley M, "Extreme Value or Trolls on Top? The Characteristics of the Most Litigated Patents" (2009), 158 *University of Pennsylvania Law Review* 1

Ang S, "The Idea-Expression Dichotomy and Merger Doctrine in the Copyright Laws in the US and the UK" (1994), 2 *International Journal of Law & Information Technology* 2, at 111-153

Arundel A, "The Relative Effectiveness of Patents and Secrecy for Appropriation" (2001), 30 *Research Policy* 4, at 611-624

Bailey R, "The Tragedy of the Anticommons: Do Patents Actually Impede Innovation?" (2007), *REASON MAGAZINE*

Bainbridge D, "Court of Appeal Parts Company with the EPO on Software Patents" (2007), 23 *Computer Law and Security Report* 2, at 199-204

Bechtold S, "Software Patents in Germany. Current Developments" (July 2000). Available at SSRN and retrieved 30.05.2012, from: http://ssrn.com/abstract=240205

Bergstra J, and Klint P, "About 'Trivial' Software Patents: The IsNot Case" (2007), 64 *Science of Computer Programming* 3, at 264-285

Bessen J, and Hunt R, "An Empirical Look at Software Patents" (2007), 16 *Journal of Economics and Management Strategy* 1, at 157-189

Burk D, and Lemley M, "Designing Optimal Software Patents", in R Hahn (eds) *Intellectual Property Rights in Frontier Industries: Software and Biotechnology* (AEI Press 2005)

Burk D, and Lemley M, "Is Patent Law Technology Specific?" (2002), *Berkeley Technology Law Journal*, Vol. 17, at 1155

Burk D, and Lemley M, "Policy Levers in Patent Law" (2003), 89 *Virginia Law Review* 7, at 1575-1696

Bärwolff M, "Beyond Copyright and Patents for Software" (2002), 9 *Computer & Society*, Technical University of Berlin Publications

Campbell-Kelly M, and Valduriez P, "A Technical Critique of Fifty Software Patents" (January 2005). Available at SSRN and retrieved 30.05.2012, from: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=650921

Canfield K, "The Disclosure of Source Code in Software Patents: Should Software Patents Be Open Source?" (2006), 7 *Columbia Science and Technology Law Review* 6

Capek P, Frank S, Gerdt S, and Shields D, "A History of IBM's Open Source Involvement and Strategy" (2005), 2 *IBM Systems Journal* 44, at 249

Carleton D, "A Behavior-Based Model for Determining Software Copyright Infringement" (1995), 10 *Berkeley Technology Law Journal* 2

Chesbrough H, "Open Innovation. The New Imperative for Creating and Profiting from Technology" (2003), *Harvard Business School Press*, at 93-112

Chesbrough H, "The Era of Open Innovation" (2003), 44 *MIT Sloan Mgmt Rev*. 35

Clarkson G, "Cyberinfrastructure and Patent Thickets: Challenges and Responses" (2007), 12 *First Monday* 6

Cockburn I, and MacGarvie M, "Patents, Thickets and the Financing of Early-Stage Firms: Evidence from the Software Industry" (2009), 18 *Journal of Economics & Management Strategy* 3, at 729-773

Cohen D, "Comment Article 69 and European Patent Integration" (1998), 92 *Nw. U. L. Rev*. 1082, at 1092-1112

Cohen J, "Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implications of 'Lock-Out' Technologies" (1994-1995), 68 *Southern California Law Review* 5, at 1091-1202

Cohen W, Nelson R, and Walsh J, "Protecting their Intellectual Assets: Appropriability Conditions and Why U.S. Manufactory Firms Patent (or Not)" (2000). NBER Working Paper Series No 7552. Retrieved 30.05.2012, from: http://www.nber.org/papers/w7552

Collins K, "An Initial Comment on In Re Bilski: Tangibility Gone Meta" (2008). Retrieved 30.05.2012, from: http://www.patentlyo.com/patent/law/collinsmetabilski.pdf

Comino S, and Manenti F, "Dual Licensing in Open Source Software Markets" (January 2010). Available at SSRN and retrieved 30.05.2012, from: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=985529

Cook W, and Lees G, "Test Clarified for UK Software and Business Method Patents: but what about the EPO?" Comments: [2007] *EIPR* 115–118

Derclaye E, "Software Copyright Law: Can Europe Learn from American Case Law?", (2000), 22 *European Intellectual Property Review* 2, at 7-16 (Part 1) and 56-68 (Part 2)

Dinwoodie G, and Dreyfuss R, "Diversifying without Discriminating; Complying with the Mandates of the TRIPs Agreement" (2007), 13 *Michigan Telecommunications and Technology Law Review* 2, at 445-456

Diver L, "Would the Current Ambiguities within the Legal Protection of Software Be Solved by the Creation of a Sui Generis Property Right for Computer Programs?" (2008), 3 *Journal of Intellectual Property Law & Practice* 2, at 125-138

Duffy J, "Rethinking the Prospect Theory of Patents" (2004), 71 *University of Chicago Law Review* 2, at 439-510

Eisenberg R, "Academic Freedom and Academic Values in Sponsored Research" (1988), 66 *TEX. L. REV*. 1363

Epstein R, "Law and Economics: Its Glorious Past and Cloudy Future" (1997), *University of Chicago Law Review*

Evans D, and Layne-Farrar A, "Software Patents and Open Source: the Battle Over Intellectual Property Rights" (2004), 9 *Virginia Journal of Law and Technology* 10

Evans D, and Reddy B, "Government Preferences for Promoting Open-Source Software: a Solution in Search of a Problem" (2003), 9 *Mich. Telecomm. Tech. L. Rev*. 313

Gabriel R, & Goldman R, "Open Source: beyond the Fairy Tales" (2002), *Perspectives on Business Innovation*, Ernst & Young, Issue 8

Ginsburg J, "Four Reasons and a Paradox: the Manifest Superiority of Copyright Over Sui Generis Protection of Computer Software" (1994), 94 *Colum. L. Rev*. 8, at 2559-2572

Guadamuz A, "Legal Challenges to Open Source Licenses" (2005), 2:2 *SCRIPT-ed*, 301-308

Guadamuz A, "The Software Patent Debate" (2006), 1 *Journal of Intellectual Property Law and Practice* 3, at 196-206

Gunther A, and Wuermeling U, "Software Protection in Germany- Recent Court Decisions in Copyright Law" (1995), 11 *Computer Law & Security Report* 1, at 12-17

Haapanen A, "The is No Such Thing as Free Lunch – Implied Patent Grant under Open Source Software Copyright Licenses" (2008), *Law in The Internet Society*

Hall B, "On Copyright and Patent Protection for Software and Databases: A Tale of Two Worlds", in Granstrand O, *Economics, Law and Intellectual Property. Seeking Strategies for Research and Teaching in a Developing Field* (2004), Kluwer Academic Publishers

Hoeren T, "The EC Directive on Software Protection- A German Comment", (July/August 1991), *Computer Law & Practice*, at 246-248

Holbrook T, "Return of the Supreme Court to Patents" (2007), 1 *Akron Intellectual Property Journal* 2

Howes A, “Disaster Pending? EPO vs English Court of Appeal on Excluded Subject Matter”, 08/07 *World Intellectual Property Report*

Johnson-Laird A, “Reverse Engineering of Software: Separating Legal Mythology from Actual Technology” (1992), 5 *Software Law Journal* 2, at 331-354

Johnson-Laird A, “Software Reverse Engineering in the Real World” (1994), 19 *U. Dayton L. Rev.* 3, at 843-902

Jordan R, “On the Scope of Protection for Computer Programs under Copyright Computer Programs: the Patent/Copyright Interface” (1989), 17 *A.I.P.L.A. Q.J.* 3, at 199-214

Karels M, “Commercializing Open Source” (July/August 2003), *QUEUE*, at 4755

Karjala D, “A Coherent Theory for the Copyright Protection of Computer Software and Recent Judicial Interpretations” (1997), 66 *U. Cincinnati L. Rev.* 1, at 53-118

Karjala D, “Distinguishing Patents and Copyright Subject Matter” (2003), 35 *Conn. L. Rev.* 2, at 439-524

Kline M, “Requiring an Election of Protection for Patentable/Copyrightable Computer Programs” (1985), 67 *J. Pat. & Trademark Off. Soc.'Y* , at 280

Kikuchi M, “Patent Eligibility and Patentability of Computer Software Patents in the United States, Europe and Japan” (2009), 16 *CASRIP Newsletter* 3

Kolle G, “Technik, Datenverarbeitung und Patentrecht—Bermerkungen zur Dispositionsprogramm—Entscheidung des Bundesgerichtshofs”, *GRUR* 1977-02, 58–74

Lakhani K, and Wolf R, “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects” (2005), in Feller J, Fitzgerald B, Hissam S, and Lakhani K, *Perspective of Free and Open Source* (2005), MIT Press

Laub C, “Software Patenting: Legal Standards in Europe and the US in View of Strategic Limitations of the IP Systems” (2006), 9 *Journal of World Intellectual Property* 3, at 344-372

Laurie R, “Use of a Levels of Abstraction Analysis for Computer Programs Computer Programs: the Patent/Copyright Interface”: Comment (1989), 17 *A.I.P.L.A. Q.J.* 3, at 232-236

Leith P, “Patenting Programs as Machines” (2007), 4:2 *SCRIPT-ed* 214. Retrieved 30.05.2012, from: http://www.law.ed.ac.uk/ahrc/script-ed/vol4-2/leith.asp

Lemley M, "Convergence in the Law of Software Copyright" (1995), 10 *Berkeley Technology Law Journal* 1, at 1-35

Lemley M, "Ignoring Patents" (3 Jul 2007). Stanford Public Law Working Paper No. 999961. Available at SSRN and retrieved 30.05.2012, from: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=999961

Lemley M, "Rational Ignorance at the Patent Office" (2000-2001), 95 *Northwestern University Law Review* 4, at 1495-1532

Lemley M, "The Economics of Improvements in Intellectual Property Law" (1997), 75 *Texas Law Review* 5, at 989-1084

Lemley M, and Brien D O', "Encouraging Software Reuse" (1997), 49 *Stanford Law Review* 2, at 255-304

Lemley M, and Cohen J, "Patent Scope and Innovation in the Software Industry" (2001), 89 *California Law Review* 1

Lemley M, and Shapiro C, "Patent Holdup and Royalty Stacking" (2007), 85 *Texas L. Rev.* 7, at 1991-2050

Lemley M, and Shapiro C, "Probabilistic Patents" (2005), 19 *Journal of Economics Perspectives* 2, at 75-98

Lerner J, and Tirole J, "Some Simple Economics of Open Source" (2000), 50 *J. Industrial Economics* 2, at 197-234

Lerner J, and Tirole J, "The Scope of Open Source Licensing" (2005), 21 *J. L. ECON. & ORG.* 20

Lerner J, and Zhu Z, "What Is the Impact of Software Patents Shifts? Evidence Form Lotus v. Borland" (2007), 25 *International Journal of Industrial Organization* 3, at 511-529

Le Stanc C, "Logiciel: Trente Ans entre Droit d'Auteur et Brevet. Bilan?" (2007), *Mélanges Linant de Bellefonds*, LexisNexis, Litec.

Lévêque F, and Ménière Y, "Copyright Versus Patents: the Open Source Software Legal Battle" (2007), 4 *Review of Economics Research on Copyright Issues* 1, at 27-46

Lin D, "Research Versus Development: Patent Pooling, Innovation and Standardization in the Software Industry" (2002), 1 *J. Marshall Rev. Intell. Prop. L.* 274

Lipton J, "IP's Problem Child: Shifting the Paradigms for Software Protection" (2006), 58 *Hasting Law Journal* 2, at 205-251

Lloyd I, “Software Patents after Fujitsu. New Directions or (Another) Missed Opportunity?”, Case Commentary (1997), 2 *The Journal of Information Law & Technology* (JILT)

Loewenheim U, “Legal Protection for Computer Programs in West Germany” (1989), 4 *Berkeley Technology Law Journal* 1

Long C, “Patent Signals” (2002), 69 *University of Chicago Law Review* 2, at 625-680

Lunney G, “Lotus v Borland: Copyright and Computer Problem” (1996), 70 *Tul. L. Rev.* 6 Part B, at 2397-2436

Magrab E, “Computer Software Protection in Europe and the EC Parliamentary Directive on Copyright for Computer Software” (1992), 23 *Law and Policy in International Business*

Mann R, “Commercializing Open Source Software: Do Property Rights Still Matter?” (2006), 20 *Harvard Journal of Law & Technology* 1

Mann R, “Do Patents Facilitate Financing in the Software Industry?” (2005), 83 *Texas Law Review* 4, 961-1030

Maurer S, and Scotchmer S, “Open Source Software: the New Intellectual Property Paradigm” (2006), *National bureau of Economic Research*

Melullis K-J, “Some Problems of Patent Law from a German Viewpoint” on *OJEPO* Special Edition 2/2007, 13th European Patent Judges Symposium, Thessaloniki 12–16 September 2006, 184–199

Menell P, “Tailoring Legal Protection for Computer Software” (1986-1987), 39 *Stanford Law Review* 6, at 1329-1373

Merges R, “Intellectual Property Rights and Bargaining Breakdown: the Case of Blocking Patents” (1994), 62 *Tennessee Law Review* 1, at 75-106

Merges R, “Software and Patent Scope: A Report from the Middle Innings” (2007), 85 *Texas Law Review* 7, at 1627-1676

Meshbesher T, “The Role of History in Comparative Patent Law” (1996), 78 *J. Pat. & Trademark Off. Soc’y* 594

Moffat V, “Mutant Copyright and Backdoor Patents: the Problem of Overlapping Intellectual Property Protection” (2004), 19 *Berkeley Technology Law Journal* 4, at 1473-1532

Mota S, “Computer Associates v. Altai- French Computer Software Copyright Action not Barred by US Decision” (1997), 3 *Journal of Technology Law and Policy* 1

Newman J, "The Patentability of Computer-Related Inventions in Europe" (1997), 19 *European Intellectual Property Review* 12

Nosko C, Layne-Farrar A, & Garcis Swartz D, "Open Source and Proprietary Software: the Search for a Profitable Middle-Ground" (2005). Available at SSRN and retrieved 30.05.2012, from: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=673861

Park J, "Evolution of Industry Knowledge in the Public Domain: Prior Art Searching for Software Patents" (2005), 2 *SCRIPT-ed* 1, at 47-70

Pila J, "Article 52(2) of the Convention on the Grant of European Patents: what Did the Framers Intend? A Study of the Travaux Préparatoires" (2005), 36 *IIC* 755

Plotkin R, "Computer Programming and the Automation of Invention: a Case for Software Patent Reform" (2003), 7 *UCLA Journal of Law and Technology* 2

Pugh A, and Majerus L, "Potential Defenses of Implied Patent License under the GPL" (2006), *Fenwick and West LLP*

Reichman J, "Legal Hybrids between the Patent and Copyright Paradigms" (1994), 94 *Columbia L. Rev*. 8, at 2432-2558

Reichman J, "Overlapping Proprietary Rights in University-Generated Research Products: the Case of Computer Programs" (1992), 17 *Columbia-VLA Journal of Law & the Arts* 1, at 51-126

Risch M, "How Can Whelan v. Jaslow and Lotus v. Borland both Be Right? Reexamining the Economics of Computer Software Reuse" (1999), 17 *John Marshal Journal of Computer and Information Law* 511, at 511-556

Rowe K, "Why Pay for what's Free?: Minimizing the Patent Threat to Free and Open Source Software" (2008), 7 *John Marshall Review of Intellectual Property Law* 3, at 595-620

Ruffin M, & Ebert C, "Using Open Source Software in Product Development: a Primer" (2004), 21 *IEEE Software* 1

Röttinger M, "Legal Protection of Computer Programs in Germany: Renunciation of Copyright?" (1987), 4 *Computer Law And Practice* 34, at 34-36

Saito K, and Sweeney R, "Assessment of Inventive Step or Obviousness in United States, Europe, and Japan", Ctr. for Advanced Study and Research – Univ. of Wash. School of Law, 3-4 (2006)

Salomon V, "Patenting Computer Software and Business Methods in the UK" (2007), *Communications Law* 12

Samuel O, "An Uneasier Case for Copyright than for Patent Protection of Computer Programs" (1993), 72 *Nebraska Law Review* 2, at 351-453

Samuelson P, "Are Patents on Interfaces Impeding Interoperability?" (2009), 93 *Minnesota L. Rev*. 1943

Samuelson P, "A Case Study on Computer Programs", in *Global Dimension of Intellectual Property Rights in Science and Technology*, Office of International Affairs, National Research Council, NATIONAL ACADEMY PRESS Washington D.C. (1993), at 284-318

Samuelson P, "Comparing U.S. and E.C. Copyright Protection for Computer Programs: Are They More Different than They Seem?" (1994), 13 *J. Law & Comm*. 297

Samuelson P, "CONTU Revisited: the Case against Copyright Protection for Computer Programs in Machine-Readable Form" (1984), *Duke L.J*. 4, at 663

Samuelson P, "IBM's Pragmatic Embrace of Open Source" (2006), *Communications of the ACM* 49 (10)

Samuelson P, "Survey on the Patent/Copyright Interface for Computer Programs" (1989), 17 *A.I.P.L.A. Q.J*. 3, at 256-293

Samuelson P, "The Story of Baker v. Selden: Sharpening the Distinction between Authorship and Invention", in J Ginsburg, and R Dreyfuss, *Intellectual Property Stories* (2005), Foundation Press

Samuelson P, "Why Copyright Law Excludes Systems and Processes from the Scope of its Protection" (2007), 85 *Texas Law Review* 7, at 1921-1978

Samuelson P, *et al*., "A Manifesto Concerning the Legal Protection of Computer Programs" (1994), 94 *Colum. L. Rev*. 8, at 2308-2431

Samuelson P, and Scotchmer S, "The Law and Economics of Reverse Engineering" (2002), 111 *Yale Law Journal* 7, at 1575-1664

Sarvas R, "More Ambiguity in European Software Patenting" (2001), Helsinki Institute for Information Technology- HIIT. Retrieved 30.05.2012, from: http://www.effi.org/patentit/sarvas_eu_patent.pdf

Schröder R, "Copyright in Computer Programs- Recent Developments in the Federal Republic of Germany" (1986), 8 *EIPR*, at 179-183

Shapiro C, "Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard–Setting" (2001), 1 *Innovation Policy and the Economy*, at 118-150

Sieber U, “Copyright Protection of Computer Programs in Germany (pts I & II)” (1984), *EIPR*, Vol. 6, at 214-253

Smith B, and Mann S, “Innovation and Intellectual Property Protection in the Software Industry: An Emerging Role for Patents?” (2004), 71 *The University of Chicago Law Review* 1, at 241-264

Soininen A, “The Software and Business Method Patent Ecosystem: Academic, Political, Legal and Business Developments in the US and Europe”, in A Soininen, *Patents in the Information and Communications Technology Sector—Development Trends, Problem Areas and Pressures for Change* (2007), Lappeenranta University of Technology

Stallman R, “The Danger of Software Patents” (2005). Retrieved 30.05.2012, from: http://notabug.com/2002/rmsessays.pdf

Stern R, “A Sui Generis Utility Model Law as an Alternative Legal Model for Software”, (1993), 1 *U. Balt. Intell. Prop. L. J.*, at 108

Tellis W, “Application of a Case Study Methodology” (1997), 3 *The Qualitative Report* 3

Tellis W, “Introduction to Case Study” (1997), 3 *The Qualitative Report* 2

Thumm N, “Blocking Patents and Their Effects on Scientific Research: Evidence from the Biotechnology Industry” (2005), *IPR Helpdesk Bulletin*, No 23

Ulmer E, and Kolle G, “Copyright Protection of Computer Programs” (1983), 14 *IIC* 2, at 159

Vivant M, and Bruguière J, “Protéger les Inventions de Demain” (2003), *La Documentation Française*, at 16

Välimäki M, “Dual Licensing in Open Source Software Industry” (2003), 8 *Systemes d’Information et Management* 1, at 63-75

Warren-Boulton F, Baseman K, and Woroch G, “The Economics of Intellectual Property Protection for Software: the Proper Role for Copyright” (1995), *EconWPA* 9411004

West J, and Gallagher S, “Challenges of Open Innovation: the Paradox of Firm Investment in Open-Source Software” (2006), 3 *R&D Management* 36, at 319-331

Weyand J, and Haase H, “Patenting Computer Program. New Challenges” (2005), *IIC* Issue 6, at 647–663

Wiebe A, "European Copyright Protection of Software from a German Perspective" (1993), 3 *Computer Law & Practice*, at 79-86

Wiebe A, "Implementation of the EC Directive on Software Protection in Germany" (1993), 1 *IT Law* 5, at 8-11

Zekos G, "Software Patenting" (2006), *The Journal of World Intellectual Property* 9, at 426–444

TREATIES AND LEGISLATION

Arrêté du 19 septembre 1979 Relatif aux Modalités de Dépôt des Demandes de Brevet d'Invention et de Certificat d'Utilité et d' Inscription au Registre National des Brevets

Bekanntmachung der Neufassung des Patentgesetzes (PatG), vom 16, Dezember 1980. Available in English at the "Collection of Laws for Electronic Access" (CLEA) of the World Intellectual Property Organization (WIPO). Retrieved 30.05.2012, from: http://clea.wipo.int

Budapest Treaty on the International Recognition of the Deposit of Microorganisms for the Purposes of Patent Procedure (28 April, 1977)

Convention on the Grant of European Patents of 05 October 1973

Council Directive of 14 May 1991 on the Legal Protection of Computer Programs (91/250/EEC)

Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the Legal Protection of Databases

Directive 98/44/EC of the European Parliament and the Council of 6 July 1998 on the Legal Protection of Biotechnological Inventions

Draft Agreement on the Establishment of a European Patent Litigation System (2004). Retrieved 30.05.2012, from: http://documents.epo.org/projects/babylon/eponet.nsf/0/B3884BE403F0CD8FC125723D004ADD0A/$File/agreement_draft_en.pdf

Draft Statue of the European Patent Court (2004). Retrieved 30.05.2012, from: http://documents.epo.org/projects/babylon/eponet.nsf/0/885CCB85F5CC33ABC125723D004B15F9/$File/statute_draft_en.pdf

Finnish Copyright Act (2010), available in English and retrieved 30.05.2012, from: http://www.finlex.fi/en/laki/kaannokset/1961/en19610404.pdf

Implementing Regulations to the Convention on the Grant of European Patents (2011). Retrieved 30.05.2012, from: http://www.epo.org/law-practice/legal-texts/epc.html

Leahy-Smith America Invents Act (AIA) (2011)

Patent Act 1977: Patentable Subject Matter, UK-Intellectual Property Office. Retrieved 30.05.2012, from: http://www.ipo.gov.uk/pro-types/pro-patent/p-law/p-pn/p-pn-subjectmatter.htm

Proposal for a Directive of the European Parliament and on the Patentability of Computer Implemented Inventions, Brussels (20 February, 2002), COM(2002)92 final

Semiconductor Chip Protection Act of 1984, title III of Pub. L. No. 98-620, 98 Stat. 3335, 3347 (November 8, 1984)

The Council of the European Union, "Draft Agreement of a Unified Patent Court and Draft Statute – Revised Presidency Text" (19 October, 2011)

The Council of the European Union, "Proposal for a Regulation of the Council and the European Parliament Implementing Enhanced Cooperation in the Area of the Creation of Unitary Patent Protection", 11328/11 (23 June, 2011)

United States Code- Title 17- COPYRIGHTS (2011)

United States Code - Title 35- PATENTS (2011)

WIPO Treaty on Trade Related Aspects of Intellectual Property Rights, Morocco (15 April, 1994)

CASE LAW

Case Law of the U.S. Courts and U.S. Patent Office

Amazon.com vs. Barnesandnoble.com (1999), U.S. District Court District Court of Seattle, No.: C99-1695P

Apple Inc., vs. High Tech Computer Corp. (2010), U.S. District Court District of Delaware, a/k/a HTC Corp., HTC (B.V.I.) Corp., HTC America, Inc., Exedea, Inc.

Apple Inc., vs. SAMSUNG ELECTONICS CO. LTD, et al. (2011), U.S. District Court California Northern District (Oakland), No.: cv-11846

AT&T Corp. vs. Excel Communications, Inc. (1999), 172, Fed. Cir. F.3d 1352

Baker vs. Selden (1879), 101 S. Ct. 99

Barracuda, Inc., vs. Trend Micro, Inc. (2007), U.S. District Court Northern District of California (San Jose Division), No.: 3:07-cv-01806-MHP

Barracuda, Inc., vs. Trend Micro, Inc. (2007), USITC, No. 337-TA-624, 72 Fed. Reg. 74, 329

Bilski vs. Kappos (2010), 08 S. Ct. 964

Blackboard Inc. vs. Desire2Learn Inc. (2006), U.S. District Court Texas Eastern District Court (Lufkin), No.: 9:2006cv00155

Blackboard Inc. vs. Desire2learn Inc. (2008), U.S. District Court Texas Eastern District Court (Lufkin), "Judgment and Permanent Injunction", No.: 9:06CV155

Caldera Sys., Inc. vs. Int'l Bus. Machs. Corp. (2003), U.S. District Court District Court of Utah, No.: 03-cv-0294

Chamberlain vs. Skylink, (2004) No.: 381 F.3d 1178

Classen Immunotherapies vs. Biogen IDEC (2011), Fed. Circ., No.: 06-1634-1649

Computer Associates, Inc., vs. Quest Software, Inc., et. al. (2004), No.: 02 C 7421

Computer Associates vs. Altai (1992), 982 F 2d 693

CyberSource Corp. vs. Retail Decisions, Inc. (2011), Fed. Circ., No.: 04-cv-03268

Diamond vs. Diher (1981), 450 S. Ct. 175

eBay vs. MercExchange (2006), 126 S. Ct. 1836

Ex Parte Bernard L. Bilski and Rand A. Warsaw (2006), BPAI, No.: WL 4080055

FireStar Software, Inc., vs. Red Hat, Inc., et al. (2006), U.S. District Court Texas Eastern District Court, No.: 2-06cv-258

Fonar Corp vs. General Electric Co. (1997), Fed. Cir. 107 F.3d 1543, 1549

Global Patent Holdings LLC vs. CDW Corp. (2007), U.S. District Court Northern District of Illinois Eastern Division, No.: 07cv4476

Global Patent Holdings LLC vs. Green Bay Packers (2008), U.S. District Court Northern District of Illinois Eastern Division, No.: 00-C-4623

Gottschalk vs. Benson (1972), 409 S. Ct. 63

In Re Abrams (1951), Fed. Cir. 188 F.2d 165, 89 U.S.P.Q. (BNA) 266

In Re Bilski (2008), Fed. Cir F.3d, No.: WL 4757110, 88 U.S.P.Q.2d (BNA) 1385

In Re Comiskey (2006), Fed. Cir., No.: 09/461, 742

In Re Xerox Corp (1975), 86 F.T.C. 364

Jacobsen vs. Katze (2008), Fed. Cir., No.: 2008-1001

KSR Intern. Co vs. Teleflex Inc. (2007), 127 S.Ct. 1727

Lexmark International vs. Static Control Components (2004), U.S. Court of Appeal for the Sixth Circuit, No.: 387 F 3d 522

LizardTech, Inc. vs. Earth Resource Mapping, Inc. (2005), Fed. Cir., 424 F.3d 1336

Lotus Development Corp. vs. Borland International Inc. (1995), U.S. Court of Appeal for the First Circuit, No.: 49 F.3d 807, 34 USPQ2d 1014

Mayo vs. Prometheus (2011), 10 S. Ct. 1150

Microsoft Corp. vs. Barnes and Noble, Inc., et al. (2011), U.S. District Court Western District of Washington at Seattle

Microsoft Corp. vs. Motorola, Inc. (2010), U.S. District Court Western District of Washington at Seattle

Microsoft Corp. vs. Tom Tom NV and Tom Tom, Inc. (2009), U.S. District Court Western District of Washington at Seattle

Northern Telecom, Inc vs. Datapoint Corp. (1990), 111 S. Ct. 296

Oracle America, Inc., vs. Google Inc. (2010), U.S. District Court California Northern District (Oakland), No.: 3:10-cv-3561

Paice LLC vs. Toyota Motor Corp. (2007), U.S. District Court Texas Eastern District Court (Marshall Division), No.: WL 2385139, 5

Palnetary Motion, Inc. vs. Techsplosion, Inc. (2001), United States Court of Appeal for the Eleventh Circuit, 261 F.3d 1188

Parker vs. Flook (1978), 437 S. Ct. 584

Plains Cotton Coop. Assoc. vs. Goodpasture Computer Serv., Inc. (1987), 108 S.Ct. 80

Progress Software Corp. vs. MySQL AB (2001), Civil Action No.: 01-11031 PBS

Red Hat vs. SCO-Caldera (2004), U.S. Federal Court District of Delaware, No.: 1-03-cv-772

SCO-Caldera vs. AutoZone (2004) U.S. District Court District Court of Nevada, No.: 2-04-cv-00237-RCJ-GWF

SCO-Caldera vs. DaimlerChrysler (2004), Oakland Country Circuit Court (Michigan), No.: 2004-056587

SCO-Caldera vs. IBM (2003), U.S. District Court District Court of Utah, No.: 2-03-cv-294

SCO-Caldera vs. Novell (2004), U.S. District Court District Court of Utah, No.: 2:04cv0139

SCM Corp. vs. Xerox Corp (1981), U.S. Court of Appeal for the Second Circuit, No.: 645 F. 2d 1195

State Street Bank & Trust Co. vs. Signature Financial Group Inc. (1998), Fed. Cir., No.: 149 F.3d 1368

Synercom Technology vs. University Computing Co. (1978), U.S. District Court Northern District of Texas (Dallas Division), No.: 462 F. Supp. 1003, 199 U.S.P.Q. (BNA) 537

Ultramercial, LLC, vs. Hulu, LLC. (2011), Fed. Circ., No.: 09-cv-6918

Wang Lab, Inc. vs. America Online, Inc. (1999), Fed. Cir., No.: 197 F.3d 1377

Whelan Associates Inc vs. Jaslow Dental Laboratory, Inc. (1987), U.S. Court of Appeal for the Third Circuit, No.: 797 F.2d 1222, 230 U.S.P.Q. 481

White-Smith Music Co. vs. Apollo (1908), 209 S.Ct. 1

z4 Technologies, Inc. vs. Microscop Corp. (2006), U.S. District Court Texas Eastern District Court, No.: 434 F. Supp. 2d 437, 440

Europe

Case Law of the EPO Boards of Appeal and EPO Enlarged Board of Appeal

Auction Method / Hitachi [2004] *OJEPO* 575 (T 0258/03)

Computer Program Product/IBM [1999] *OJEPO* 609 (T 0935/97)

Computer Programs Product/IBM [1999] *OJEPO* 609 (T 1173/97)

Computer Related Invention/Vicom [1987] *OJEPO* 14 (T 0208/84)

Controlling Pension Benefit System/PBS Partnership [2001] *OJEPO* 441 (T 0931/195)

EPO Case Law of the Boards of Appeal, Headnote of opinion G 3/08. Retrieved 30.05.2012, from: http://www.epo.org/law-practice/legal-texts/html/caselaw/2010/e/clr_headwords.htm

EPO Enlarged Board of Appeal Decision G 2/95 (21 December, 1994) and G 2/98 (21 May, 2001)

EPO Opposition Hearing Regarding "Amazon gift ordering patent" EP 0927945 B1 (07 December, 2007)

Koninklijke Philips Electronics N.V. [2007] (T 1191/04)

Microsoft/ Data transfer expanded clipboard formats [2006] (T 0424/03)

Text processing/IBM [1989] *OJEPO* 118 (T 0038/86)

Two Identities/COMVIK [2002] *OJEPO* 7/2003, 352 (T 641/00)

X-Ray Apparatus/Koch & Sterzel [1987] *OJEPO* 585 (T 0026/86)

Case Law of the Courts and Patent Office in the United Kingdom

Acres Gaming Incorporated [2005] EWHC 2416

Astron Clinica Ltd and Others vs. Comptroller General [2008] EWHC 85 (Pat)

Autonomy Corporation Limited vs. Comptroller General [2008] EWHC 85 (Pat)

Cantor Fitzgerald International vs. Tradition (UK) Ltd [1999] Masons CLR 157

Cappellini/Bloomberg [2007] EWHC 476 (Pat)

CFPH LLC [2005] EWHC 1589

Court of Appeal (England and Wales), *Aerotel Ltd vs. Telco* [2006] EWCA Civ 1371

Fujitsu Limited's Application [1997] RPC 608, [1997] EWCA Civ 1174

Gale [1991] RPC 305

Halliburton Energy Services, Inc., vs. Smith International [2005] EWHC 1623

Ibcos Computers Ltd vs. Barclays Mercantile Highland Finance Ltd [1994] FSR 275

John Richardson Computers Ltd vs. Flanders and Chemtec Ltd [1993] FSR 497

Navitaire Inc. vs. EasyJet Airlines Co [2004] EWHC 1725 (Ch)

Nova Production vs. Mazooma Games and Ors [2007] EWHC Civ 219

Re Oneida Indian Nation [2007] EWHC 0954, (Pat)

Symbian Ltd vs. Comptroller General [2008] EWHC 518 (Pat)

Case Law of the Courts and Patent Office in the Germany

Antiblockiersystem, BGH GRUR 1980, 849

Coditel vs. Cine Vog Films (C-62/79) [1980] E.C.R. 1-881

Dispositionsprogramm, BGH GRUR 1977, 96–98. Available in English in 8 *Int'l Rev. Indus. Prop. & Copyright L.* 558, 560 (1977)

Deutsche Grammophon (C- 78/70) [1971] E.C.R. 1-487

Inkassoprogramm, BGH (German Supreme Court) (May 09, 1985), 87 GRUR 1041-1047. Available in English at: (1986) 17 *IIC* 81, at 681

In Re Welte vs. D-Link Germany, District Court of Frankfurt am Main, No.: 2-6 0 224/06 (22/09/2006)

In Re Welte vs. Sitecome Germany, District Court of Münhen I , No.: 21 0 6123/04 (19/05/2004)

Logikverifikation, BGH GRUR 2000, 498–499

Operating System- Betriebssystem, BGH (04/10/1990), Case No.: I ZR 139/89. Available in English at: (1991) 22 *IIC* 5, at 723

Rote Taube, BGH GRUR 1969, 672. Available in English in 1 *Int'l Rev.Indus. Prop. & Copyright L.* 136 (1970)

Show Format- Sendeformat, BGH 2003, Case No.: I ZR 176/01. Available in English at: (2004) 35 *IIC* 8, at 987

Sprachanalyseeinrichtung II, BGH GRUR 2000, 1007. Available in English in *IIC* Vol. 33 No. 3/2002, at 343

Tauchcomputer, BGH GRUR 1992, 430

Welte vs. Skype Technologies SA, Higher Regional Court of Munich (08/05/2008)

Case Law of the Courts and Patent Office in the France

Agent Judiciaire du Tresor et Trace vs. Softmax, Paris Court of Appeals, 4th ch. (31/05/1995), [1995] Expertises 311

CA Paris 15 Juin 1981, PIBD 1981.185.III.175, Dossiers Brevets

CA Paris 22 Mai 1973, PIBD 173. 107.III.197, Comm. 28 Mai 1975, PIBD 1975.155.III

Computer Assoc. Int'l vs. S.A.R.L. Faster, No. 519/95, Tribunal de Commerce Bobigny (20/01/1995)

ESI vs. Mecalog, Paris Commercial Court (22/11/1993), [1994] Expertises 32

Marben GL vs. Cao Diffusion, Meaux Commerciale Court (17/12/1996), [1997] Expertises 122

Simci vs. Digimedia, Paris Court of Appeals, 4th ch. (16/02/1994), [1995] Expertises 240

Sybel Informatique vs. Oxalead and Prolepse, Tribunale de Grande Instance de Paris, 3d ch., 2d sect. (12/11/1992), [1993] Expertises 109

Paris Court of Appeal, 4th ch. A. (23/11/1994), [1995] Experises 36

Case law and Opinions of the Court of Justice of the European Communities

Advocate General Opinion C-406/10 on *SAS Institute Inc. vs World Programming Ltd.* (2011)

Avis 1/09 of the European Court of Justice, "Creation of a United Patent Litigation System" (8 March, 2011)

Case C-406/10 *SAS Institute Inc. v World Programming Ltd* JUDGEMENT OF THE COURT (Grand Chamber) (2 May, 2012), [2012] ECR 0

WEBSITES

Retrieved 30.05.2012.

Fedora Project, at http://fedoraproject.org/

Foundation for a Free Information Infrastructure e.V. (FFII), at: http://ffii.org/

Free and open source movements, at: http://endsoftwarepatents.org

"Free Software Definition", GNU Operating System, at: http://www.gnu.org/philosophy/free-sw.html

Freshmeat, at: http://freshmeat.net/about

GPL-violations.org, at http://www.gpl-violations.org/

Groklaw, at: http://www.groklaw.net/

Harald Welte's blog, at: http://laforge.gnumonks.org/weblog/

Open-IP, at: http://www.open-ip.org

Open Source As Prior Art (OSAPA) project, at: https://www.linux-foundation.org/en/Osapa

Open Source Initiative, at: http://www.opensource.org/

Oracle's MySQL Blog, at: http://blogs.oracle.com/MySQL

Patent Common project, at: http://www.patentcommons.org/

PatExpert, at: http://www.patexpert.org

Prior aRT and Software Patents project, at:
http://wiki.mozilla.org/Legal:Prior_Art#Summary

Sourceforge, at: http://sourceforge.net

The Linux Foundation, at: http://www.linuxfoundation.org/

FOSS LICENSES

Retrieved 30.05.2012.

Apache License, version 2.0, at: http://www.apache.org/licenses/LICENSE-2.0.html

Common Public License, Version 1.0, at: http://www.eclipse.org/legal/cpl-v10.html

GNU General Public License, version 3, at: http://www.gnu.org/copyleft/gpl.html

Mozilla Public License, version 2.0, at: http://www.mozilla.org/MPL/2.0/

Open Source Licenses, at: http://www.opensource.org/licenses

OTHER MATERIAL

EPO, Guidelines for Examination (2007)

European Patent Office, Annual Report 2004

European Patent Office, Annual Report 2007

European Software Patent Statistics, Retrieved 30.05.2012, from:
http://eupat.ffii.org/patents/stats/index.en.html

EU Commission, Internal Market, Patent Litigation Insurances Studies, Retrieved 30.05.2012, from:
http://ec.europa.eu/internal_market/indprop/patent/index_en.htm

Federal Trade Commission, “To Promote Innovation: the Proper Balance of Competition and Patent Law and Policy”. *A Report by the Federal Trade Commission* (October, 2003)

IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology (1990), New York, The Institute of Electrical and Electronics Engineers

IPR Helpdesk, Patentability of Computer Programs (2005), Retrieved 30.05.2012, from: http://www.ipr-helpdesk.org

Manual of Patent Practice Guidance for Interpreting the Patent Act 1977, Retrieved 30.05.2012, from: http://www.ipo.gov.uk/pro-types/pro-patent/p-law/p-manual/p-manual-practice/p-manual-practice-pat1977.htm

"Patents for Software? European Law and Practice" (2009). Retrieved 30.05.2012, from: http://documents.epo.org/projects/babylon/eponet.nsf/0/a0be115260b5ff71c125746d004c51a5/$FILE/patents_for_software_en.pdf

Referral under 112(1)b) EPC by the President of the EPO (Patentability of programs for computers) to the Enlarged Board of Appeal, pending under Ref. N° G3/08 (23 Oct, 2008)

Report of the President's Commission on the Patent System, "To Promote the Progress [...] of the Useful Arts", in an Age of Exploding Technology, 13 (1966)

"Research Use of Patented Knowledge: a Review", STI Working Paper 2006/2, OECD Directorate for Science, Technology and Industry (STI)

Réponse du Groupe Français de l'Association Internationale pour la Protection de la Propriété Industrielle (AIPPI) à la Consultation Établie par les Services de la Direction Générale du Marché Intérieur de la Commission Européenne: Brevetabilité des Inventions Mises en Oeuvre, Rapport Q 133. Retrieved 30.05.2012, from: http://www.industrie.gouv.fr/observat/innov/carrefour/aippi.htm

Schutz von Computerprogrammen. Retrieved 30.05.2012, from: http://www.dpma.de/patent/patentschutz/schutzvoncomputerprogrammen/index.html

Tang P, Adams J, and Paré , "Patent Protection of Computer Programs", Final Report (2001) (Submitted to European Commission, Directorate General Enterprise)

United States Patent and Trademark Office, 2007-2012 Strategic Plan. Retrieved 30.05.2012, from: http://www.uspto.gov/web/offices/com/strat2007

"Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defence" by MITRE Corporations, Version 1.2.04 (2003)

USPTO Written Description Training Material (25 March, 2008)

"World Patent Report. A Statistical Review", WIPO (Ed. 2008)

APPENDIX 2 INDIVIDUAL ESSAYS

ESSAY I (Single authored) **"**Scope of IP Protection for the Functional Elements of Software", Published in *In Search of New IP Regimes* (2010), Publications of IPR University Center, pp. 27-62 (Peer reviewed)

ESSAY II (Single authored) "Software Patents in Europe. The Technical Requirement Dilemma", *Journal of Intellectual Property Law & Practice* 2008(3)9:563-575 (Peer reviewed)

ESSAY III (Single authored) "The Software Patent Thicket: A Matter of Disclosure" (2009) 6:2 *SCRIPTed* 207 (Peer reviewed)

ESSAY IV (Single authored) "Proprietary Software vs. FOSS: Challenges with *Hybrid* Protection Models" IPR University Center Publications, IPR Series B:4 (2012) (Peer reviewed)

ESSAY I

Rosa Maria Ballardini

Scope of IP Protection for the Functional Elements of Software

Published in *In Search of New IP Regimes* (2010), Publications of IPR University Center, pp. 27-62

Peer reviewed

Minor modifications have been made to the original publication and its formatting.

Scope of IP Protection for the Functional Elements of Software

Rosa Maria Ballardini[*]

Abstract

It is well-known that the pluralistic nature of computer programme code has made it challenging to keep up the distinction between copyrightable and patentable aspects of software. The dual nature of software, as both literary work of art and performing (practical) functions at the same time, has been particularly problematic for intellectual property laws.

This article argues that both European copyright and patent laws urgently need some refinement in order for them to interact with each other in a way that would promote rather than impede innovation in such a fundamental field of technology as computer software. The more pressing questions concern the scope to be accorded to copyright and patents, particularly with respect to the protection of software's most valuable assets, namely its functional elements. It is evident that when it comes to functional aspects, patents are a much better protection mechanism than copyright. However, in order to maintain the distinction between copyrightable and patentable objects in software two rules should be considered.

Firstly, on the copyright front, this article advocates the adoption of explicit statutory limitations against non-literal infringement to keep the scope of software copyright protection within the sphere of literary works of art.

Secondly, the paper suggests that keeping software within the umbrella of patent law is essential in order to impede copyright from expanding to protect the functional elements of software in a 'patent-like' manner.

* Researcher in IP law at HANKEN School of Economics; member of the INNOCENT Graduate School in IP law, IPR University Center, University of Helsinki, rosamaria.ballardini@hanken.fi. The author would like to thank Professor Niklas Bruun, Assistant Professor Marcus Norrgård and Professor Pamela Samuelson for their valuable comments. The usual disclaimer applies.

1. INTRODUCTION

Copyright and patents are the most common intellectual property mechanisms used to afford legal protection to computer programmes. The scope of copyright, however, differs vastly from the scope of patents.

Generally speaking, copyright is designed to protect literary, original works of authorship from copying, for a period of seventy years after the author's death.[1] The scope of copyright is defined by the rather vague 'idea-expression dichotomy', under which copyright protects expressions of ideas and should not extend to the ideas *per se*.[2] To be protected by copyright a certain work requires a sufficient level of creativity or originality. Patents, on the other hand, give the right to exclude others from making, using, or selling the patented art for a period of twenty years from the filing date.[3] Patents protect the implementation of ideas and apply to functional, non-literary objects. The scope of patent protection is defined by the claims.

From a theoretical perspective, if a single creation includes both protectable ideas and protectable expressions, each regime of IP law should apply to different, mutually exclusive aspects of the creation. Usually, in fact, it is easy to distinguish between literary and non-literary elements of a certain work.[4] However, although distinguishing between literary and non-literary is simple for creations in which the expression of ideas and the ideas themselves can be fully separated from each other, problems arise when the expression is an integral part of the idea itself and, thus, expression and ideas are inseparable.[5] This is often the case with software.

Precisely, the aspect defining the problem is that the programme's code has a pluralistic nature, possessing inseparable literary and functional elements. This suggests that software constitutes a 'hybrid' object that, as such, deviates from those traditionally protected by intellectual property laws.[6]

This article focuses on the IP protection for the software functional aspects.[7] Even though utilitarian elements should not fall under the umbrella of copyright law, the difficulty of drawing the boundary between literary and functional aspects in software has sometimes extended copyright to cover functions, thereby protecting software in a 'patent-like' manner. Extending copyright law to protect functional works of technology, however, not only contravenes both copyright's traditional principles and explicit statutory provisions, but might also be highly detrimental for software innovation due to copyright's broad scope of protection and long term of duration.

---

1 See Council Directive of 14 May 1991 on the legal protection of computer programs (91/250/EEC) (from now on: *Software Copyright Directive*); 17 U.S.C. (2000).

2 *Software Copyright Directive* Art. 1.2 and Recital 14; 17 U.S.C. § 102(b) (2000).

3 European Patent Convention (EPC), Art. 63 ; 35 U.S.C., Chapter 10.

4 For example, with a work of literature, 'literal' elements comprise the words, sentences and paragraphs as expressed in print, while 'non-literal' elements consist of the detailed plot, sequence of events, characters and scenes. See D Bainbridge, *Intellectual Property* (Longman, 2006), at 245-250.

5 See also O Samuel, "An Uneasier Case for Copyright than for Patent Protection of Computer Programs", 72 *Nebraska Law Review* 2, at 351-453 (1993).

6 See J Reichman, "Legal hybrids between the patent and copyright paradigms", (1994) 94 *Columbia L. Rev.* 8, at 2432-2558.

7 The fact that some of the most difficult issues relate to software code, does not mean that all digital age IP questions focus on program's code. The legal protection of graphical user interfaces (GUIs), for instance, is a highly controversial and discussed problem. This discussion, however, goes beyond the scope of this paper.

Over the years, courts have been uneven in affording copyright protection to computer programmes. American jurisprudence abounds with decisions addressing the issue. After having searched for the perfect solution for many years, under current rules, the functional aspects of programmes are granted a relatively narrow level of protection in the US. In Europe, instead, the traditional reluctance to afford patent protection to computer programmes under the dictate of Article 52(2)(3) of the European Patent Convention (EPC), has lead to a system that tends to favour a broad scope of software copyright. On one hand, the Software Copyright Directive leaves open possibilities for broad interpretations of the scope of software copyright at the national level. On the other, national rulings have, thus far, been scarce and inconsistent.[8] As a consequence, European courts are still far from developing a uniform line of interpretation by which to judge these issues.

The present article kicks off with a critique of the 'idea-expression' dichotomy for software copyright. The long history of software copyright in the US, as well as the advanced development of the country's jurisprudence, serve as a background for discussion on the European situation.

Specifically, this paper argues that both European copyright and patent laws are responsible for most of the still-open questions surrounding the legendary dichotomy. To be able to interact with each other in a way that promotes, rather than impedes, innovation in such a fundamental field of technology as computer software, both sets of legal rules urgently need some refinement. The more pressing questions revolve around the scope to be accorded to copyright and patent protection. Precisely, two rules should be considered.

First, the copyright scope should be narrowed down to protect merely against literal infringement. The need to maintain non-literal copying of the code under the umbrella of copyright law is highly questionable under current law. Additionally, confining copyright to literal infringement would ensure that copyright rules only apply to the software's expressive elements without extending to its functions.

Second, maintaining software patents is important in order to provide an adequate level of protection to the most valuable aspects of software, i.e. its functional elements. This might also be the only workable manner in which to maintain the distinction between copyrightable and patentable objects in software. The highly debated question of software as a patentable subject matter in Europe (and, currently, also in the US) is, thus, missing the real point. The software sector would, instead, benefit much more from the concentration of these valuable efforts into an amelioration of the current, imperfect patent system.

2. SOFTWARE IS A PLURALISTIC WORK

Software is a pluralistic work possessing several elements, each of which could fall into different categories of IP laws. Computer programmes can be defined as 'a combination of computer instructions and data definitions that enable computer hardware to perform

8 See E Derclaye "Software Copyright Law: Can Europe Learn from American Case Law?", (2000) 22 *European Intellectual Property Review* 2, at 7-16 (Part 1) and 56-68 (Part 2).

computational or control functions'.[9] IP protection applies differently according to the manner in which those instructions and definitions are expressed.

This article assesses legal rules with specific concern for the functional elements of software. Therefore, it is important to point out from the beginning what is to be considered 'functional' and what, instead, should be taken as 'expressive' in software.

2.1. Functional Aspects of Computer Programmes

The development of most computer software comprises several stages. *Task definition* (that is, describing the task to be accomplished) usually represents the initial phase. Immediately following is *flowcharting*, which consists of depicting schematically the way in which the programme will accomplish the task. *Coding* then consists of converting the elements of the flowchart into source code and object code.[10] While coding, the programmer first asserts the set of instructions in the form of source code.[11] In order to be 'understood' by the hardware, however, the source code needs to be translated into object code,[12] that is, machine-readable instructions. *Debugging* and writing the *documentation* usually represent the last two steps of the programme-writing process. Debugging involves testing the programme for accuracy, correcting programming errors, and verifying that the programme functions properly. Finally, documenting consists of preparing materials that explain the functioning of the programme.[13]

Computer software is composed of both a computer programme (i.e. source code and object code) and a series of ancillary materials, such as documentation (e.g. specifications and maintenance documentation), computer files or data, etc. Here, the focus is placed on computer programmes. Issues related to functionality, in fact, mostly arise with respect to computer programmes, while it is unquestionable that specifications, manuals and documentation are purely literary works.[14]

At first glance, computer programmes appear to be textual work. Source code is expressed in written form; thus, it seems logical to define it as literary work and, as such, protectable under copyright law.[15] However, while conceiving of programmes as texts is not incorrect, it is definitely incomplete.

The most important characteristic of computer programmes is the fact that they 'behave': the importance of programmes lies in that they provide instructions to make hardware perform certain tasks.[16] Behaviour is not a secondary aspect of computer programmes, but rather an essential part of what programmes are. To put it in other

---

9 See IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology, 1990, New York, The Institute of Electrical and Electronics Engineers.

10 P Menell, "Tailoring Legal Protection for Computer Software", (1986-1987) 39 *Stanford Law Review* 6, at 1329-1373.

11 Source codes are lines of instructions written in some human-readable computer language. See note 9 above.

12 *Ibid.*

13 See Menell, note 10 above.

14 This paper gives the same meaning for both the term 'computer program' and 'software' in the context of IP protection. Although the actual significance of 'software' and 'computer program' differs, in fact, these terms are often used interchangeably under the assumption that the difference between the two is clear to all parties in the discussion.

15 Object code constitutes purely a 'translation', directly resulting from the source code; thus, its legal status is usually considered indistinguishable from that of the source code.

16 P Samuelson et al, "A Manifesto Concerning the Legal Protection of Computer Programs", (1994) 94 *Colum. L. Rev.* 8, at 2308-2431.

words: no one would purchase a software that did not function, no matter how elegant or creative the programme's code might be.

A programme's functionality consists of everything that happens once the programme is executed on the target system (e.g. a general computer).[17] Among the functions commonly found in word processing programmes, for instance, are: copying text, deleting text, moving text from one place to another and aligning margins.[18]

The demarcation line between literary and functional aspects in programmes seems relatively pervious in theory. In practice, however, such a border has proved much less simple to assess. This is clearly shown by some of the most significant rulings passed down in the field, as will be explained later.

2.2. Literary v. Functional

The fact that lines of programming code are not only literary, but also perform a function creates some obvious problems for copyright law.

It should be pointed out that computer programmes are not the only literary works that also possess functional aspects or serve utilitarian purposes. Choreographies, musical scores, and stage direction are similar examples.[19] However, for most of these works, the performative, that is functional, manifestation of the work directly depends upon their literary manifestations.[20]

Computer programmes are unique in that there is a significant degree of independence between literary and functional manifestations. The software functions, in fact, are fully independent from the grammatical, i.e. literary, construction of the lines of code. In other words, even though the source codes of two programmes might look completely different, such codes can perform the exact same function, producing the same (or a very similar) set of instructions. As pointed out already in 1994 by the authors of the Manifesto, computer programmes are 'machine[s] whose medium of construction happens to be text'.[21]

The fact that programme behaviour in general is not protectable by copyright law because of its functionality does not mean that such behaviour can never be protected by copyright. Sometimes programme behaviour can be considered 'expressive' in a traditional copyright sense and, thus, copyright applies.[22] Distinguishing when programme behaviour is expressive and when it is functional, however, is very challenging. The difficulty arises, in particular, when assessing the non-literal copying of certain aspects, elements, and/or levels of a copyrighted programme.[23] As a consequence, copyright protection has occasionally been extended to software's utilitarian elements, protecting functional ideas in a 'patent-like' manner.

---

17 See also D Carleton, "A Behavior-Based Model For Determining Software Copyright Infringement", (1995) 10 *Berkeley Technology Law Journal* 2.

18 See note 16 above.

19 See J Ginsburg, "Four Reasons and a Paradox: the Manifest Superiority of Copyright Over *Sui Generis* Protection of Computer Software", (1994) 94 *Colum. L. Rev.* 8, at 2559-2572.

20 See D Carleton, "A Behavior-Based Model For Determining Software Copyright Infringement", (1995) 10 *Berkeley Technology Law Journal* 2.

21 See note 16 above.

22 See P Samuelson, "Why Copyright Law Excludes Systems and Processes from the Scope of Its Protection", (2007) 85 *Texas Law Review* 7, at 1921-1978.

23 G Lunney, "Lotus v Borland: Copyright and Computer Problem", (1996) 70 *Tul. L. Rev.* 6 Part B, at 2397-2436.

3. COPYRIGHT FAILURES

For many years, software has been primarily protected by copyright law and considered a literary work. International and regional treaties such as the TRIPS Agreement, the US Copyright Act and the European Community Software Copyright Directive are confirmation of generally accepted copyright protection.

The application of traditional copyright rules, however, has never been easy. Defining the scope of copyright protection has been a main cause of concern. In particular, software's pluralistic nature has often made it impossible to draw the line between those parts of programmes that should be regarded as literary and creative and those that, instead, should be considered mechanical or functionally dictated.[24]

Software functionality is not the only problem with copyright protection. Other issues complicate the software copyright environment under current rules. Some, like the long duration of copyright protection in relation to the short lifespan of software and the challenges involved with the definition of 'originality' of the code, are old problems. Some others have derived from new technological developments. Professor Lipton, for instance, has highlighted some of the difficulties created by the increased use of modularisation and object-oriented designs in computer programming.[25]

The fact that these problems are as old as the adoption of copyright for software does not mean that the issues addressed in and affected by them have been solved. 'Boundary' concerns, in particular, remain highly controversial. While the literature in the field is extensive, the question has not yet found any workable solution, and the dilemma is still deeply entrenched in the system. This is particularly true in Europe, where courts have not yet been able to develop precise guidelines for the interpretation of software copyrightability.

On these grounds, this article derives the impellent need to take a step back and look at this debate again from its roots. Specific attention is placed on the problems derived from the dual, literary-functional nature of computer programmes. In this respect, probably the two more controversial issues relating to copyright are the failure of copyright law to protect the most valuable parts of the code (i.e. its functional elements) on one hand, and the difficulty of drawing the boundary between functional and expressive elements in computer programmes on the other.[26]

Even though exposing as well as finding solutions for the other mentioned concerns (e.g. long copyright duration, originality, and technical concerns) is certainly very important, such discussion goes beyond the scope of this paper and, thus, is intentionally left for further investigation.

3.1. The Pluralistic Nature of Software: Risks of Under-Protection

Software source code has traditionally been considered a literary work in most legal systems,[27] mainly due to the fact that such code is expressed 'in writing'. As mentioned

24 See note 16 above.

25 See J Lipton, "IP's Problem Child: Shifting the Paradigms for Software Protection", (2006) 58 *Hasting Law Journal* 2, at 205-251.

26 See note 16 above.

27 See 17 U.S.C. §102(a) (1988); *Software Copyright Directive*, Art. 1; see also TRIPs Agreement, Art. 10.

earlier, copyright merely extends to the expression of the programme in the form of either source or object code, but does not afford protection to the way the programme works.

The difficult part in producing a piece of software, however, lies in understanding the problem, i.e. in developing the idea, rather than writing the code. Once the problem has been discerned and solved, in fact, writing a different code is a fairly simple thing to do.[28] In other words, the fact that copyright does not protect companies from sharing their technical information, allows competitors to understand the way their products work and reproduce them without infringement.[29]

In the US, this shortcoming of copyright has partly been filled by the introduction of software patent protection. In Europe, the extension of patent law to computer-implemented inventions (CII) has also provided a partial answer to this problem. By impeding competitors from writing code that includes any patented aspect of the software implementation, patents constitute the most efficient IP mechanism for blocking or, at least, making it difficult to duplicate programme's functionality.[30] In this way, the ready availability of software-related patents plays a major role in reducing the risks of software copyright under-protection. Both the recent trend towards the restriction of patentable subject matter in the US and the fact that the EPC affords only limited protection to software-related inventions might, however, compromise this achievement.

It is very important to keep this in mind when discussing the interaction of IP rules in software. In particular, the limitation of patentable subject matter should be considered and evaluated in view of the consequences that the extension of patent protection to computer-related inventions brings to the entire IP software framework.

3.2. Drawing the Boundary: Risks of Over-Protection

A fundamental principle of copyright law is that unauthorised copying is not permitted.[31] Unauthorised copying not only includes the literal appropriation of words and expressions used by the author (i.e. literal copying), but also the appropriation of the essence of the author's expression without using the author's actual words (i.e. non-literal copying). As mentioned earlier, the pluralistic nature of software codes has rendered the assessment of non-literal infringement particularly challenging. How should literary and functional features be distinguished in software? Is copyright infringed if the functional parts of the software code are copied?

Over the years, courts have struggled to provide a clear-cut answer to these questions. This is particularly evidenced by the extensive American jurisprudence. Notwithstanding the much less abundant case law, this same challenge has clearly emerged on the European front as well.

---

28 R Mann, "Do Patent Facilitate Financing in the Software Industry?", (2005) 83 *Texas Law Review* 4, at 961-1030.

29 R Jordan, "On the Scope of Protection for Computer Programs under Copyright Computer Programs: The Patent/Copyright Interface", (1989) 17 *A.I.P.L.A. Q.J.* 3, at 199-214.

30 *Ibid*.

31 *Software Copyright Directive*, Article 4-6-7; 17 U.S.C. § 106(1) (2000).

The main difficulty is that when only concepts and not actual language are copied, the courts are put to the challenging test of having to distinguish the idea from its expression.[32] Precisely these aspects represent the bases of the often-discussed 'idea-expression dichotomy'. The difficulty in applying such a dichotomy to the software context leads to risks of copyright over-protection.

3.3. The Idea-Expression Dichotomy

The US Copyright Act and the EU Software Copyright Directive have both incorporated the rather vague idea-expression dichotomy to determine the scope of copyright protection.[33] The dichotomy originates from the basic principle that copyright should not protect abstract ideas or functional objects, but rather the expression of such ideas.[34] On the contrary, ideas and principles should be left free for anyone to use. The dichotomy, thus, aims at delineating with a sufficient degree of precision the demarcation line between the 'ideas' sealed in an invention and the 'expression' of those ideas.[35]

Even though the dichotomy applies similarly to each type of artistic work, both the dual literary-utilitarian nature and the complexity of computer programme codes have contributed to making the application of such a doctrine particularly troublesome in the software context.[36]

It should also be noticed that the risks of over-protection (in the same way as the risks of under-protection) have been reduced in the US by the introduction of patents for software. The clear availability of patents, in fact, has made the extension of copyright to the structural and functional elements of computer codes less important.

In Europe, instead, the fact that restrictions have been placed on the patentability of software has led to a more generous system in terms of copyright. As is discussed in the next section, it can, generally speaking, be said that the more a country favours the patentability of software-related inventions, the more reluctant it is to extend copyright to the functional aspects of software.

3.4. Relevant Jurisprudence

European and American legislations converge on issues related to the idea-expression principle. In each system, however, the law remains general and very broad; thus, court decisions and guidelines are essential to clarify their meanings.

The idea-expression dichotomy is very well developed under American copyright law and its application has had interesting effects on software protection. After having searched

---

32 The American Copyright Act excludes every kind of 'ideas, procedures, processes, systems, methods of operation, concepts, principles or discoveries' from copyright protection, while the EU Software Copyright Directive states that 'to the extent that logic, algorithm and programming languages comprise ideas and principles, those ideas and principles are not protected under this Directive'.

33 See note 8 above.

34 See, for instance, *Baker v. Selden* 101 U.S. 99 (1879).

35 S Ang, "The Idea-Expression Dichotomy and Merger Doctrine in the Copyright Laws in the US and the UK", (1994) 2 *International Journal of Law & Information Technology* 2, at 111-153.

36 See P Samuelson, "The Story of Baker v. Selden: Sharpening the Distinction Between Authorship and Invention", in J Ginsburg, and R Dreyfuss (eds), *Intellectual Property Stories*, (Foundation Press, 2005). Available at SSRN: http//ssrn.com/abstract=743545 (accessed 14 Sept 2009).

for a solution to the scope of copyright protection for software for several years, American courts have now established a fairly extensive body of well-settled case law addressing the issue. Overall, it is commonly accepted that copyright should provide a very 'thin' level of protection for software and should in no case extend to the programmes' structure, sequence and organisation (SSO).

In Europe, on the contrary, the case law on software copyrightability is not yet very abundant. As a result, the few decisions addressing the issue have not yet provided a satisfactory answer to the question of the proper copyright scope of protection for computer programmes.[37] Developing a uniform, consistent interpretative line for judging these issues, however, is necessary in order to avoid inconsistency and legal insecurity.

3.4.1. United States

3.4.1.1. First Decisions- Contrasting Lines of Interpretation

The difficulty of assessing non-literal infringement in software copyright has driven various courts to search for precise, applicable rules. In the US, most questions on the interpretation of copyright law for computer programmes were asked during the 1980s, immediately after the extension of copyright protection to software.

The first approach used to deal with non-literal infringement of programmes was very simple: courts rejected copyright protection for the non-literal copying of software code. An example of such a trend is the *Synercom Technology v. University Computing Co.*[38] decision.

Synercom had designed 'a series of *input formats* which would accept data from users in conjunction with a programme which analysed engineering problems with the design of buildings'.[39] Defendant EDI developed a programme using another programming language which accepted information in the same formats as Synercom's programme. The court held that the sequencing and ordering of data input constituted the idea, rather than the expression, of the Synercom programme. Accordingly, the court found no infringement in the copying of those sequences.

The same conclusion was endorsed also in *Plain Cotton Cooperative Ass'n Inc. v. Goodpasture Computer Service*.[40] In this case, the Fifth Circuit held that the structure and organisation of computer input formats could not be protected because they were determined in large part by functional considerations.

This initial approach was soon considered too simplistic. In particular, it was argued that this practice would have also led to the denial of copyright protection for obviously expressive elements of a programme if the infringer copied the 'look and feel' of those

37 See note 8 above.

38 *Synercom Technology v University Computing Co.* 462 F. Supp. 1003 (N.D. Tex. 1978).

39 M Lemley, "Convergence in the Law of Software Copyright", (1995) 10 *Berkeley Technology Law Journal* 1, at 1-35.

40 *Plain Cotton Cooperative Ass'n Inc. v. Goodpasture Computer Service*, 807 F.2d 1256 (5th Cir. 1987).

elements, instead of their actual implementation, into the code.[41] A completely opposite interpretative line came from a subsequent decision in the *Whelan v. Jaslow*[42] dispute.

The appeal court was called upon to determine the scope of copyright protection for a custom computer programme used for record-keeping in a dental laboratory. Whelan had developed a computer programme to the specifications of its customer, Jaslow. Subsequently, Jaslow designed and began marketing its own programme in competition with Whelan, which sued for infringement. Jaslow had not copied Whelan's code, but rather its overall structure, sequence and organisation (SSO).

In making the decision, the Third Circuit expanded the scope of copyright protection far beyond the literal copying of executable code. Under the *Whelan* test, in fact, copyright extends to software's SSO as long as the overall purpose of the entire programme, broadly defined, can be implemented in another manner. On these bases, the *Whelan* court reasoned that because a function could be performed in more than one way, its SSO was to be considered expressive and, thus, copyrightable.

It is not difficult to imagine that the *Whelan* decision sparked a firestorm of criticism for over-expanding software copyright. Specifically, there were two key flaws in the *Whelan* court test. First, merely asking whether another way of performing a particular function exists does not resolve the idea-expression dilemma. Second, the decision extended copyright to incorporate things traditionally unprotected under copyright rules, such as facts, elements dictated by efficiency or functional considerations, patentable processes and expression which has 'merged' with the underlying idea.[43] Only the 'central idea' was explicitly left out of copyright protection.

3.4.1.2. The Altai Doctrine

The wave of criticism derived from Whelan led to a series of decisions that radically narrowed down the scope of software copyright.[44] *Computer Associates v. Altai* [45] initiated this trend.

Computer Associate (CA) sued Altai for copying parts of its operating system scheduling programme to make Altai's programme compatible with CA's. CA claimed that the new Altai software infringed its copyright in the non-literal elements of the programme.

In deciding in favour of Altai, the Second Circuit endorsed what has since became known as the 'abstraction, filtration, comparison' test (AFC test) for judging copyright infringement. The AFC test is a three-step approach that relies on a system of analytic

41 See note 39 above.

42 *Whelan Associates Inc v Jaslow Dental Laboratory Inc* [1987] FSR 1.

43 These elements were later considered as unprotectable under copyright law in the Gates Rubber decisions. See *Gates Rubber Co. v Bando Chem. Indus.*, 9 F.3d 823,834 (10th Circ. 1993).

44 See, for instance, *Synercome Technology, Inc. V University Computing Co.*, 474 F. Supp. 37 (N.D.Tex.); *Plains Cotton Coop. Assoc. v. Goodpasture Computer Serv., Inc*, 807 F.2d 1256 (5th Cir. 1987) reh'g denied en banc, 813 F.2d 407 (5th Cir. 1987), cert. denied, 108 S. Ct. 80 (1987).

45 *Computer Associates v Altai* 982 F 2d 693 (1992).

dissection in order to break the programme into its constituent parts and then evaluate the copyrightability of each individual section.[46]

The first step of the AFC test attempts to break the structure of the programme into different levels, from the most abstract to the most detailed.[47] The second step seeks to 'filter', at each level of abstraction, protectable elements from those that cannot be protected. Ideas are obviously left out of the group of protectable elements. The last step of the test is infringement analysis: courts should compare the elements remaining after the filtration stage in order to evaluate issues of substantial similarity.

3.4.1.3. Recent Developments

With the establishment and common acceptance of the *Altai* approach, the courts seem now to have decided what test to use when it comes to distinguishing utilitarian from non-functional elements and, thus, address computer programme copyright issues. The difficult formulation and application of the *Altai* doctrine, however, represents yet another umpteenth failure in the unfortunate story of software copyright law. The complexity of the debate as well as the problems faced with the application of the AFC test, in fact, clearly indicate that courts remain fundamentally uncertain of how broadly to demarcate software copyright protection.[48]

Certainly, the application of the AFC test has acquired greater sophistication and more precision on its way to general acceptance. Nevertheless, there are still a number of fundamental problems inherent with the modern applications of copyright law in terms of drawing the line between copyrightable and non-copyrightable code. For instance, although text book descriptions clearly suggest that under current rules software obtains only a narrow scope of protection, some recent case comments indicate that this might not always be the case in practice. On the contrary, questions pertaining to the appropriate scope of copyright protection for software do not always have clear answers.

Chamberlain v. Skylink[49] and *Lexmark International v. Static Control Components*[50] are two such examples. Both cases involved 'lock out' codes, that is, software that operated as a security system attached to a device (e.g. printer or computer) to bar the use of unauthorised components along with that product.[51] Neither court was particularly convinced that the code in question was copyrightable. However, the judges were clearly split on questions related to the copyrightability of the code.[52]

On one hand, it appears clear that the US courts generally agree that software should be afforded a narrow scope of protection under copyright law. On the other, the practical

---

46 For more information on the 'abstraction-filtration-comparison' doctrine, see R Laurie, "Use of a Levels of Abstraction Analysis for Computer Programs Computer Programs: The Patent/Copyright Interface": Comment (1989) 17 *A.I.P.L.A. Q.J.* 3, at 232-236.

47 For more details see note 39 above.

48 See note 22 above. See also M Risch, "How Can *Whelan v. Jaslow* and *Lotus v. Borland* Both Be Right? Reexamining the Economics of Computer Software Reuse", (1999) 17 *John Marshal Journal of Computer and Information Law* 511, at 511-556. Available at SSRN: http://ssrn/abstract=885341 (accessed 16 Nov 2009).

49 *Chamberlain v Skylink*, 381 F.3d 1178; 2004 U.S.App. LEXIS 18513 (2004).

50 *Lexmark International v Static Control Components*, 387 F 3d 522 (6th Cir. 2004); 2004 U.S.App. LEXIS 22250 (2004).

51 See note 10 above.

52 See note 25 above.

application of these principles might not be that insignificant and a more precise conception of what should be excluded by copyright law might still be necessary.

3.4.2. Europe

The EU Software Copyright Directive provides a potentially broad (or 'thick') level of protection for the underlying structure of a computer programme.[53] One possible reason for such a configuration might be the traditional European reluctance to afford software patent protection. The EPC excludes computer programmes 'as such' from patent law.[54] Allegedly, this might have brought the legislator to leave the doors open for a potential expansion of copyright protection.[55]

Because the Directive *per se* is very general, its interpretation through case law is necessary in order to develop a uniform approach. In contrast with the plentiful amount of software litigation in the United States, though, European courts have not been faced with a tremendous number of cases on the issue of the scope of copyright protection.

Next, the leading jurisdictions of the UK, Germany and France are taken as examples to demonstrate that courts across Europe have not yet developed any precise test for assessing the copyrightability of computer programmes. There is reason to believe that this trend is shared also by other European jurisdictions.

3.4.2.1. United Kingdom

The First Cases- A Generous Approach

In addressing the copyrightability of computer programmes, the British courts have taken a cautious approach by clearly pointing out that copyright protection extends to more than mere literal copying.

The traditional British requirements of 'skill and labour' (to assess copyrightability) and 'copying of a substantial part of the work' (to assess infringement) have led to a system that generally supports quite a broad level of copyright protection for software.[56] Additionally, this trend might also be the consequence of the well-known reluctance of British courts to extend patent protection to computer programmes.[57]

Three important decisions taken before the Software Copyright Directive entered into force particularly highlight this trend. Namely: *John Richardson Computers Ltd. v. Flanders*[58], *Ibcos v. Barclays*[59] and *Cantor Fitzgerald International v. Tradition*.[60]

---

53 P Samuelson, "Are Patents on Interfaces Impeding Interoperability?", (2009) 93 *Minnesota L. Rev.* 1943. See also P Samuelson, "Comparing U.S. and E.C. Copyright Protection For Computer Programs: Are They More Different Than They Seem?", (1994) 13 *J. Law & Comm.* 297.

54 See European Patent Convention, Article 52 (2)(3).

55 *Ibid.*

56 See note 8 above.

57 See R M Ballardini, "Software Patents in Europe: the Technical Requirement Dilemma", (2008) 9 *Journal of Intellectual Property Law & Practice* 3, at 563-575.

58 *John Richardson Computers Ltd v Flanders and Chemtec Ltd* [1993] FSR 497.

59 *Ibcos Computers Ltd v Barclays Mercantile Highland Finance Ltd* [1994] FSR 275 case.

Richardson v Flanders was the first British case to address the issue of non-literal copying in software. While working at Richardson Computers, Mr. Flanders had developed a programme to be used by pharmacists to produce labels for prescriptions and for stock control. When Flanders came up with a new version of the programme for IBM personal computers, claimant sued him for copyright infringement and breach of confidence.

In deciding the case, Judge Ferris aimed at applying the AFC American test. Notwithstanding his intentions, Ferris ended up using the traditional British requirements of copyrightability and infringement, and ruled in favour of the claimant on grounds of substantial similarity.

Ibcos v. Barclays was the first British case to consider issues of literal copying in computer programmes.[61] Of particular relevance is the comparison made by Jacob between British and American copyright laws and the critique made concerning the application of American precedents in the UK. Jacob argued that American copyright specifically excludes functional works from copyright protection, while in the UK, such exclusion has never been formulated. As a consequence, in the UK, a 'detailed idea' might be protected by copyright.

Finally, in *Fitzgerald v. Tradition*, Pumfry elaborated further from *Ibcos*, and reaffirmed that the structure of a programme is protected by copyright law if it shows sufficient 'skill and labour'.

Narrowing Down the Scope of Copyright Protection

Recently, two important decisions, based on the EU Software Directive, seem to have narrowed down the scope of software copyright protection in the UK.

Navitaire v. EasyJet[62] showed the English High Court's reluctance to follow the criticisms of US authority in *Ibcos*. The case referred to an action for copyright infringement brought by Navitaire against EasyJet and BulletProof Technologies, over airline reservation software. No access to the source code of the allegedly copied software could be proved. Instead, BulletProof emulated the operation and functionality of the programme by carefully studying its use (e.g. seeing how it behaved, what functions and operations it could perform, how it manipulated, stored and retrieved information, etc.).

Navitaire sued for infringement of its copyright by 1) using the look and feel of its software, 2) detailed copying of the many commands entered by the user to achieve particular results, and 3) copying screen displays and reports displayed on the screen. Questions about whether the structure and functionality of the code had been copied, whether there was non-textual copying of the so called 'look and feel' of the programme and whether what Navitaire was trying to protect lay on the wrong side of the idea-expression dichotomy were raised.

Pumfrey held that, in view of the software Directive, any extension of copyright protection to the matters presented by Navitaire would have been inappropriate and unjustified. Specifically, 'what is left when the interface aspects of the case are

60 *Cantor Fitzgerald International v Tradition (UK) Ltd* [1999] Masons CLR 157.

61 D Bainbridge, *Intellectual Property* (7th eds. 2008), at 245-247.

62 *Navitaire Inc v EasyJet Airlines Co* [2004] EWHC 1725 (Ch).

disregarded is the business function of carrying out the transaction and creating the record'. This was not relevant skill and labour.

Agreeing with the *Navitaire* ruling, in *Nova Production v. Mazooma Games and Ors*[63] the Court of Appeals confirmed that:

1. The making of a computer programme which emulates another programme but which does not copy the other programme's code or graphics, is not an infringement of copyright;

2. Ideas which underlie the programme are not protected by copyright.

These two decisions placed some limitations on the scope of software copyright in the UK, launching an approach that, to a certain extent, might reduce the possibility of broad copyright protection for computer programmes.

3.4.2.2. Germany

After an initial controversy over whether the fundamental nature of copyright could be applied to 'technical' works like computer programmes,[64] software copyright slowly began to be commonly accepted by German courts.[65]

In Germany, however, the discussion on software copyrightability has mostly evolved around the degree of 'individuality' (or 'creativity') required.[66] By the mid-1980s German case law was refusing copyright protection for computer programmes based on the very high threshold of creativity required under the sever *Überdurchschnitlichkeit* test.[67] Both in the *Inkassoprogramm*[68] and in the *Operating System- Betriebssystem*[69] cases, for instance, the courts rejected copyright protection for computer programmes by requiring a significant amount of creativity with respect to the 'selection, accumulation, arrangement and organisation' of the programmes.

63 *Nova Production v Mazooma Games and Ors* [2007] EWHC Civ 219.

64 See U Loewenheim, "Legal protection for computer programs in West Germany", (1989) 4 *Berkeley Technology Law Journal* 1. See also Schröder, "Copyright in Computer Programs- Recent Developments in the Federal Republic of Germany", (1986) 8 EIPR, at 179-183.

65 See E Ulmer and G Kolle, "Copyright Protection of Computer Programs", (1983) 14 *IIC* 2, at 159; Sieber, "Copyright Protection of Computer Programs in Germany (pts I & II)", (1984) 6 *EIPR* , at 214-253.

66 See A Wiebe, "European Copyright Protection of Software from a German Perspective", (1993) 3 *Computer Law & Practice*, at 79-86. See also A Gunther and U Wuermeling, "Software Protection in Germany- Recent Court Decisions in Copyright Law", (1995) 11 *Computer Law and Security Report* 1, at 12-16.

67 See note 65 above.

68 *Inkassoprogramm*, BGH (German Supreme Court) 09.05.1985, 87 GRUR 1041-1047. Available in English at: (1986) 17 *IIC* 81, at 681. See also A-A Wandtke, W Bullinger, *Praxiskommentar Zum Urheberrecht* (München: C. H. Beck, 2006) § 69a, 22-31; A Gunther and U Wuermeling, "Software protection in Germany- Recent court decisions in copyright law", (1995) 11 *Computer Law & Security Report* 1, at 12-17; M Röttinger, "Legal protection of computer programs in Germany: renunciation of copyright?", (1987) 4 *Computer Law And Practice* 34, at 34-36.

69 *Operating System- Betriebssystem*, BGH 04.10.1990, Case No. I ZR 139/89. Available in English at: (1991) 22 *IIC* 5, at 723. See also Wandtke & Bullinger and Gunther & Wuermeling note 68 above.

It should be noted, though, that since the ratification of the EU Directive the creativity requirement has been lowered accordingly.[70]

No German case has yet addressed the idea-expression distinction in the software field. However, under German copyright law, the principle of discernable creation assures that copyright protection is given only when the work is observable by other humans in some way and, thus, ideas that have not materialised from their creators' mind are not protected by copyright.[71]

This dogma was reiterated by the Federal Court of Justice (*Bundesgerichtshof* - BGH) in the *Sendeformat* case.[72] Even though the decision did not involve copyright protection of computer programmes, the principles derived from it are worthy of mention, as they are relevant also in the software context. The BGH decided that the idea underlying a TV show (TV format) was not protected by copyright because copyright protects only the result or expression of an individual activity of the mind. In this way, the court confirmed that neither the ideas, nor the methods, should be afforded copyright protection under German law.

3.4.2.3. France

In France, courts have not yet elaborated any structured test for assessing the copyrightability of computer programmes. Instead, they have relied on expert opinions in order to determine the similarity of programmes.

Cases like *Sybel Informatique v. Oxalead and Prolepse,*[73] the *Agent Judiciaire du Tresor et Trace v. Softmax*[74] and the *Marben GL v Cao Diffusion,*[75] for instance, show how experts generally have applied doctrines and standards in ways very similar to the filtration stage of the AFC test.[76]

In *Computer Assoc. Int'l v. S.A.R.L. Faster*[77] (which involved the same parties as Altai's American case) the French court adopted the same reasoning as the US court (though without specifying that it had to do so).[78]

Indeed, some French courts apply quantitative rather than qualitative tests to assess infringement.[79] Overall, however, no uniform guidelines are followed in France under current legislation.

---

70 See A Wiebe, "Implementation of the EC Directive on Software Protection in Germany", (1993) 1 *IT Law* 5, at 8-11. See also T Hoeren, "The EC Directive on Software Protection- A German Comment", (July/August 1991) *Computer Law & Practice,* 246-248; E.B. Magrab, "Computer Software Protection in Europe and the EC Parliamentary Directive on Copyright for Computer Software", (1992) 23 *Law and Policy in International Business*.

71 *Ibid*.

72 *Show Format- Sendeformat*, BGH 26.06.2003, Case No. I ZR 176/01. Available in English at: (2004) 35 *IIC* 8, at 987.

73 See *Sybel Informatique v Oxalead and Prolepse*, Tribunale de Grande Instance de Paris, 3d ch., 2d sect., November 12, 1992 [1993], Expertises 109 ; Paris Court of Appeal, 4th ch. A., November 23, 1994 [1995] Experises 36.

74 See *Agent Judiciaire du Tresor et Trace v Softmax*, Paris Court of Appeals, 4th ch., May 31 1995 [1995] Expertises 311.

75 See *Marben GL v Cao Diffusion*, Meaux Commerciale Court, December 17, 1996 [1997] Expertises 122.

76 See note 8 above.

77 *Computer Assoc. Int'l v. S.A.R.L. Faster*, No. 519/95 , Tribunal de Commerce Bobigny, January 20, 1995.

78 S Mota, "Computer Associates v Altai- French Computer Software Copyright Action Not Barred By US Decision", (1997) 3 *Journal of Technology Law and Policy* 1.

3.4.3. Some Consequences

The above analysis shows how courts both in the US and in Europe have been uneven in affording copyright protection to computer programmes.

The long US experience in the field has led to a better developed, more accurate approach to interpreting the idea-expression dichotomy of computer programmes. As seen, the establishment of the AFC test has clarified that under US law a 'thin' level of protection is to be afforded to computer software. Nevertheless, recent decisions demonstrate that even though more general questions on the scope of software copyright have been settled in the US, assessing non-literal infringement remains controversial, as American courts are still divided on the test's practical applications.

In Europe, on the other hand, courts have either not yet been fully confronted with software copyrightability issues or have decided them without structured analysis. On occasion, courts have considered the approaches taken in the US, particularly the AFC test. However, no European jurisdiction has yet developed a clear, precise line to assess the copyrightability of computer programmes. Thus, confusion remains as to which method should be adopted in order to find for infringement in a case of non-literal copying.

The present article aims at defining a workable model for European courts when dealing with questions of software copyrightability. The well-developed American jurisprudence provides a good point of comparison as well as a good starting point for the formulation of precise principles in Europe.

Thus far, at least two conclusions can be drawn from the European situation. On one hand, European trends show that a positive answer to the question of whether copyright protection might extend to the functional aspects of software code cannot be categorically excluded. On the contrary, a broad protection that extends to software functions and SSO, and that might, thus, invade the traditional province of patents by protecting 'implementations of ideas' rather than their 'expression' might easily be a possibility in the European context. This risk is particularly high in those jurisdictions, like the UK, that have been more reluctant to extend patent protection for software-related inventions. Germany and France have, instead, traditionally accorded patent protection to computer-implemented inventions, following an approach that very much resembles the one followed at the EPO.[80] Accordingly, there is reason to believe that these two countries are more willing to accord a narrow level of copyright protection to software.

On the other hand, the legal uncertainty derived from the fact that courts have been unable to formulate generalised copyrightability rules for software might create problems for companies operating in the field. Not only would it be wasteful for companies (or software developers in general) to negotiate the use of software elements that might be freely reused without a license, but also the fear of a lawsuit could create a 'chilling effect' causing authors to reuse less than might actually be permitted.[81] Unclear

---

79 See, for instance, *ESI v Mecalog*, Paris Commercial Court, November 22, 1993 [1994] Expertises 32 and *Simci v Digimedia*, Paris Court of Appeals, 4th ch., February 16th, 1994 [1995] Expertises 240.

80 For more information on the matter see note 57 above.

81 M Lemley, and D O' Brien, "Encouraging Software Reuse", (1997) 49 *Stanford Law Review* 2, at 255-304.

property rights increase also the number and costs of disputes, potentially excluding small competitors from the market.[82]

To restore confidence and legal security, therefore, specific guidelines and principles should be formulated. This is precisely the aim of the next section.

4. A MORE WORKABLE APPROACH

The analysis conducted thus far has demonstrated that copyright law, when applied to software, struggles to reach the goals it was originally designed to meet. This configuration has led many scholars and commentators to think of copyright as an inappropriate mechanism for promoting progress in the software field.

Some have advocated the abrogation of copyright protection for software.[83] Some have also suggested that, because the most valuable parts of software lie in its utilitarian side, patent law would be a more adequate protection mechanism. At the same time, others have proposed abolishing patents on abstract concepts like computer programmes altogether.[84] Finally, acknowledging the unique nature of programmes and the general inability of traditional IP laws to afford a suitable level of protection to this highly important field of technology, some academics (most notably Professor Samuelson and the co-authors of the Manifesto) have also supported the adoption of a completely different body of laws specifically tailored to accommodate the distinctive characteristics of software (i.e. a *sui generis* type of protection).[85]

This article suggests that the relevant question is not whether the existing IP framework constitutes a perfectly-fitted system to accommodate special needs of software, but rather how such a system might be better framed in order for it to serve its fundamental purpose and promote progress in the software field. In other words, the core issue lies in the scope to be accorded to both copyright and patents in order to find proper balance between protection and the dissemination of information. Efforts should thus concentrate on the amelioration of the existing legal system, rather than on seeking radical changes, such as the abrogation of all, or some, of the IP mechanisms currently used to afford software legal protection.

Specifically, this article argues for refinement of the interpretation of both copyright and patent laws in order to achieve the important goal of enhancing development and progress in the software sector. Even though focus is mainly placed on Europe, some of the recommendations proposed here might also be applied to the American context.

---

82 *Ibid.*

83 See, for instance, Lipton, note 25 above.

84 Free and open source movements. See: http://endsoftwarepatents.org (accessed 16 Nov 2009).

85 See note 16 above. See also A Guadamuz, "The Software Patent Debate", (2006) 1 *Journal of Intellectual Property Law and Practice* 3, 196-206.

4.1. Some Remarks on a *Sui Generis* Type of Protection

Before going into detail about the model this article suggests, it is necessary to briefly go through the proposal for a *sui generis* type of protection that has so long been discussed, particularly within the academia.[86]

The renowned difficulty to fit software within the framework of traditional IP rules certainly makes the option of creating a new body of laws, specifically tailored to meet the special needs of software, a highly tempting solution. First, however, satisfactory answers must be provided for a series of fundamental questions.

Would the new *sui generis* right for software eliminate existing copyright and/or patent protection? Or would these three mechanisms interact with each other? If a co-existing system were chosen, how would the interfaces between the *sui generis* protection and the copyright and/or patent protection be defined?

The answer to the question of whether the *sui generis* right should totally override existing IP mechanisms (mainly copyright and patents) would most probably be negative.

On one hand, the clear, international acceptance of copyright protection shows the extent to which this mechanism is entrenched in the system.[87] Furthermore, copyright still serves some fundamental purposes. Among these, for instance, are the fact that copyright offers a close substitute for businesses, such as SMEs, that cannot afford the highly expensive and time consuming process of obtaining a patent;[88] that software which, though it might have been created through substantial effort, does not meet the patent system's high requirements would be left unprotected because of the unavailability of copyright;[89] and that copyright is at the foundation of one of the most successful protection mechanisms in the software sector: the open source software model.[90]

On the other hand, notwithstanding the fact that the application of patent law to software is still a highly controversial topic, patent protection for computer-related inventions exists (at least to a certain extent) both in the US and in Europe; it seems unlikely that this will change in the near future. Moreover, the utilitarian elements of software are surely better protected by patent law than copyright.

The only option, therefore, would be a system where a *sui generis* right cooperates with the other protection mechanisms already in use. The vision of the interrelation between *sui generis* protection, copyright and patent rights has evolved over the years, and different models have been put forward by their supporters.[91] However, thus far, no

---

86 *Ibid*.

87 See note 19 above.

88 M Kline, "Requiring an Election of Protection for Patentable/Copyrightable Computer Programs", (1985) 67 *J. Pat. & Trademark Off. Soc.Y* , at 280; See P Tang, J Adams, and D Paré, *Patent Protection of Computer Programs. Final Report* (2001) (Submitted to European Commission, Directorate General Enterprise); P Samuelson, "Survey on the Patent/Copyright Interface for Computer Programs", (1989) 17 *A.I.P.L.A. Q.J.* 3, at 256-293.

89 See note 61 above.

90 See Open Source Licenses, at: http://www.opensource.org/licenses (accessed 16 Nov 2009).

91 See note 16 above. See also R Stern, "A *Sui Generis* Utility Model Law as an Alternative Legal Model for Software", (1993) 1 *U. Balt. Intell. Prop. L. J.*, at 108; L Diver "Would the current ambiguities within the legal protection of software be solved by the creation of a *sui generis* property right for computer programs?", (2008) 3 *Journal of Intellectual Property Law & Practice* 2, at 125-138.

common precise model has been agreed upon. Additionally, it is hardly demonstrable whether a *sui generis* system would provide clear lines at the interfaces of copyright and patents. On the contrary, intuitively, this system might further complicate the already unclear situation of the interrelationships of all the multiple protection mechanisms currently used to afford software protection. What is certain, is that the transaction costs associated with the continuous formulation of proposals for new forms of legal protection for software do not only increasingly shift resources from innovation to litigation, but it also undermine the purposes for which legal incentives are initially provided.[92]

The creation of an industry-specific body of IP laws also raises concerns as to whether such an approach should be considered for every technology that does not fit into the traditional IP framework. Software might well be exemplary of the failure of IP law to shape itself to meet the arising needs of new technologies, but it is definitely not the only case. Biotechnology and nanotechnology are but two of many fields where traditional IP law has proven insufficient.

Special *ad hoc* bodies of laws have already been adopted for different types of technology in the past. The protection of databases in the EU[93] and the protection of semiconductor chips in the USA[94] are probably the best-known examples.

To fully model IP laws on an industry-specific basis, however, would not only be difficult, but it would also not necessarily be a good option. Moulding IP law in many, different ways in relation to the technology to which they apply would create uncertainty and confusion. Furthermore, to draw the line between technologies would be extremely challenging, as a significant percentage of inventions fall into more than one field, and new fields arise regularly.[95]

Therefore, notwithstanding the possible advantages the adoption of a *sui generis* regime for software could bring, the drawbacks involved suggest that this might not be a workable option. Probably more desirable, would be the implementation of some of the suggestions put forward by the advocates of the *sui generis* proposal in the interpretation of the currently existing systems of copyright and patent rights.

4.2. Narrowing the Scope of Protection of Software Copyright

Copyright is not an appropriate tool to protect functional elements in general, and functional aspects of software in particular. Notwithstanding all the exposed problems, copyright is not always ineffective or disadvantageous for software. On the contrary, copyright still plays a crucial role in the industry's ability to appropriate returns from the innovation that it produces.

---

92 J Reichman, "Overlapping Proprietary Rights in University-Generated Research Products: The Case of Computer Programs", (1992) 17 *Columbia-VLA Journal of Law & the Arts* 1, at 51-126.

93 Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases.

94 Semiconductor Chip Protection Act of 1984, title III of Pub. L. No. 98-620, 98 Stat. 3335, 3347 (adding chapter 9, Title 17, United States Code, to provide design protection for semiconductor chips), November 8, 1984.

95 D Burk, and M Lemley, "Is Patent Law Technology-Specific?", (2002) 17 *Berkeley Tech. Law Journal* 4, at 1155-1206. Available at SSRN: http://ssrn.com/abstract=349761 (accessed 16 Nov 2009).

This suggests that legislative action to fully abrogate software copyright would not only be extremely difficult to reach, but also highly undesirable. Instead, this situation indicates that clearer rules on the scope of software copyright urgently need to be formulated.

Specifically, although copyright should continue to protect against literal copying of programme code, whether such a mechanism should be retained to protect against non-literal copying is highly questionable under current legislation.

4.2.1. Literal Copying

Typically programmers are confronted with two different types of dangers: literal and non-literal copying.

The copying of a computer programme is literal when the programme code itself is copied. In this case the two programme codes are written in the same computer programming language.

Non-literal copying, instead, occurs when elements of a computer programme such as, for instance, its structure, sequence of operation, functions, interfaces and methodologies are copied, but the programme code *per se* is not directly copied. In this case, the two programmes might be written in the same language or in different programming languages.[96]

Unlike literal, non-literal copying might not only include the copying of the expressions of ideas, but also the copying of the ideas (i.e. functional aspects) *per se*. Here, the focus has been placed on literal copying, while non-literal copying issues are discussed in detail in the next section.

Copyright is very efficient in deterring, preventing, as well as enforcing sanctions in event of the literal infringement of software code.[97] Literal software infringement is also known as 'piracy'.

Under the TRIPs Agreement, pirated copyright goods are defined as 'any goods which are copies made without the consent of the right holder or person duly authorised by the right holder in the country of production and which are made directly or indirectly from an article where the making of that copy would have constituted an infringement of a copyright or a related right under the law of the country of importation'.[98] In other words, piracy refers to a situation where third parties make wholesale, literal copies of a programme and then further copy and distribute these 'pirate' copies in a manner that suppresses demand for the genuine product from the original developer.[99]

Computer programmes are exceptionally easy to pirate, and require little skill to do so. This is particularly true when, as it is usually the case, software is distributed in the form of object code. On one hand, developing a programme is often quite costly; on the other,

96 See note 61 above.

97 See note 17 above. See also P Samuelson, and S Scotchmer, "The Law and Economics of Reverse Engineering", (2002) 111 *Yale L.J.* 7, at 1575-1664: "Copyright law protects programs from the cheapest and most rapid way to make a directly competing identical product, namely, copying program code exactly".

98 TRIPS Agreement, Section IV, Article 51, Note 14b).

99 See note 17 above.

such a cost is independent from the number of copies distributed. As duplicating and distributing the programme is usually inexpensive, once developed, the marginal cost of software production is relatively low.[100] Consequently, without copyright protection, programme codes could easily be copied in large numbers at roughly no cost.[101]

The fact that third parties might free ride on the original work without development costs increases the risk that developers might be unable to earn a competitive return on their investments in the marketplace. This might radically diminish or even eliminate inventors' incentives to invest in future innovations.[102]

Copyright is thus a very important protection mechanism against literal copying of code. However, it is not readily clear whether and to what extent the non-literal copying of programmes should be kept within the umbrella of copyright protection under current rules. This is discussed in the following section.

4.2.2. Non-Literal Copying

Unlike piracy, non-literal copying involves replicating the functionality or appearance of the original programme through the use of different or independently developed programme codes.[103]

This article has shown how the assessment of non-literal infringement in software copyright has not only led to uncertainty, as courts have followed many different and inconsistent approaches to address the issue, but also to a possible extension of copyright law to cover functions. The interdependence of the functional and literary construction of the lines of code, in fact, makes it challenging to make such an evaluation.

On one hand, these challenges have been remarkably reduced in the USA by the *Altai* doctrine. The AFC test, in fact, has established a 'thin' level of protection for software copyright. This being so, however, the test's practical application has proven to be highly challenging. Thus, judicial misunderstanding of the interpretation of non-literal infringement still remains widespread.

In Europe, both the broad dictate of the Software Directive (which leaves open possibilities for broad interpretations of the scope of software copyright at national level), and the scarce and inconsistent case law, urgently call for the development of precise guidelines to assist European judges dealing with questions of infringement.

Legal instability is certainly not the only factor that might suggest the need of a policy against protecting non-literal copying in software copyright. From an economic perspective, various studies have questioned the necessity of extending copyright for

---

100 F Warren-Boulton, K Baseman, and G Woroch, "The Economics of Intellectual Property Protection for Software: the Proper Role for Copyright", (1995) *EconWPA* 9411004, available at: http://elsa.berkeley.edu/~woroch/softcopy.pdf (accessed 16 Nov 2009).

101 See D Karjala, "A Coherent Theory for the Copyright Protection of Computer Software and Recent Judicial Interpretations", (1997) 66 *U. Cincinnati L. Rev.* 1, at 53-118.

102 B Smith, and S Mann, "Innovation and Intellectual Property Protection in the Software Industry: An Emerging Role for Patents?", (2004) 71 *The University of Chicago Law Review* 1, at 241-264.

103 It should be noted that simply re-writing the code by using a different programming language constitutes a 'translation'. In the same way, the object code is the direct result of the source code and should arguably be linked to its fate.

software much beyond the protection of literal copying, except for new, useful and unobvious improvements.

Professor Menell, for instance, opined long ago that, due to the relatively low cost of developing programmes, their generally short life span and the advantages derived from introducing the product first, the lead time needed to recover development costs may be relatively short.[104] Evidently, the lead time needed will depend on how long it takes imitators to reverse engineer the original programme. Both the complexity of computer programmes and the fact that programmes are mostly released only in object code form, ensure a significant lead time (compared to the product life) even if the IP rights protect only against literal copying.[105] Therefore, the need for strong protection against non-literal copying might not be justified.

Additionally, the way in which improvements are made in the field of computer programming suggests that a 'thin' level of copyright protection, which excludes non-literal copying, is preferable. It is not only that innovation in the programming field is highly incremental, but also that relatively small, obvious changes in the code might bring about very different functions; vice versa, very different-looking codes might easily produce the same functions. Given that the most valuable aspects of software are indeed its functions, a strong copyright protection, which extends beyond the literal copying of codes, is likely to inhibit the most dynamic sequential innovations of the programme.[106]

Finally, from a policy perspective, the coexistence of copyright and patent also raises questions about the scope of protection for each one of these IP forms. This dual legal regime is evidently the result of historical accidents, rather than of prudent policy measures unambiguously designed to improve public welfare.[107] As a consequence, there are various concerns derived from the fact that both types of protection might be used on the same software codes, but these concerns might not have been taken into account by legislators. In particular, this protection practice might deprive the public of many of the benefits that would otherwise result from the granting of either a patent or copyright only. This might disrupt both the copyright and patent balance, thereby distorting the entire IP system.

In-depth analysis of the question of coexistence is not within the scope of this paper. However, some of the problems created by such a system are worth mention.

Among the restrictions that patent-copyright coexistence creates, is the possible limiting of the right to make improvements on the patented programme.[108] If the author's exclusive right to make (or control the making) of programme codes applies not only to mere literal copying, but also to substantially similar modifications of the codes or to derivative works (i.e. non-literal copying of the code), when both copyright and patent protection exist on a certain software, copyright might limit the users' right to make improvements on the patented article. In fact, even if patent law permitted such activity, under copyright law the 'improved' version (of the programme behind the invention) is

104 See note 10 above.

105 *Ibid*.

106 *Ibid*.

107 M Bärwolff, "Beyond Copyright and Patents for Software", (2002) 9 *Computer & Society, Technical University of Berlin Publications*. Available at: http://ig.cs.tu-berlin.de/oldstatic/ap/mb/papers/Baerwolff-Beyond-2002.pdf (accessed 16 Nov 2009).

108 V Moffat, "Mutant Copyright and Backdoor Patents: the Problem of Overlapping Intellectual Property Protection", (2004) 19 *Berkeley Technology Law Journal* 4, at 1473-1532.

likely to be considered either substantially similar to the patented one, or a derivative work. In other words, the copyright owner might get the right to prevent others from copying the 'improved' code, even if this includes novel and nonobvious or inventive improvements that would otherwise be patentable. Therefore, if copyright also protects against the non-literal copying of code, it might prevent later inventors from developing and profiting from further improvements, in this way jeopardising some of the most essential goals of patent law.

4.2.3. The Proposal

The above analysis shows that even though copyright is essential for protecting against and preventing the literal copying of code, legal, economic, technical as well as public policy concerns do not seem to support its role in protecting against non-literal copying.

It should be noted that this article does not suggest that the copyright scope of programme codes should be reduced by raising the level of originality required. The reason why the criterion of 'originality' is set low in computer programme code is because code is all about efficient functioning. If, for instance, the code needed to be expressed in a more convoluted manner to pass the originality test for copyrightability, programmers could end up wasting time on writing codes that are more expressive, but that do not function more efficiently than 'less original' alternatives.[109] Accordingly, approaches like the one followed in Germany before the EU Directive entered into force and under which a very high level of creativity was required for the software code to be accorded copyright protection, do not seem justified and, as such, should not be encouraged.

Instead, what this article suggests is that the scope of copyright should be kept narrow by not extending protection beyond the literal copying of software code.[110] On one hand, it has been shown that if copyright is used to protect more than the literal code, it often ends up protecting too much. On the other, if copyright is used to protect no more than the literal code, it might protect too little. To find a proper balance and ensure that software developers will earn competitive returns on their investments, other protection mechanisms should be put in place.

When copyright was adopted as a measure of protection for software it was not really clear what role patents, trade secrecy, contracts and DRM measures would play in the software field. Nowadays, however, the picture appears much clearer. Under current legislation, for instance, DRM measures, used in combination with trade secrecy and contracts, provide effective protection to the computer code. Additionally, patents are more appropriate than copyright to protect the ideas implemented in programmes.

The proposed system would certainly not be more disadvantageous for SMEs or, in general, programmers that currently rely on copyright to protect their software. As for those programmes that, even though requiring a considerable amount of work, do not qualify for patent protection, programmers would still have a sufficient (and, arguably, more efficient) variety of protection mechanisms available.

---

109 See Guadamuz, note 85 above.

110 The article merely focused on the computer programs' code. Thus, the proposed solution would merely apply to software source code. This is clearly not to say that copyright should not be a viable form of protection for other elements of computer software, notably various outputs and documentation.

Scaling back the scope of software copyright to merely the literal copying of code would not only alleviate some major legal concerns, but would also allow copyright law to 'do its job' properly by protecting the types of works to which it is best suited, that is, literary works. On the other hand, this model would allow attention to be directed to more appropriate avenues of legal and technological protection for codes, such as appropriately tailored patent protection and sophisticated DRM measures combined with trade secrecy and contractual licensing.

These are certainly not perfect solutions. In particular, there are some major flaws in patent law, which urgently needs some refinement, as will be discussed in the next section. The model here proposed, however, would constitute an important step towards reaching a generally more coherent system of legal protection in the software context.

The scaling back of software copyright should be carefully planned in order to assure that only non-literal copying of code is taken out of the scope of copyright law. This work should not be left to the courts when a dispute arises. Instead, legislative action would be necessary in order to look more broadly at the problems arising from the functional nature of software and not focusing only on specific issues which arise in particular cases.

This article aims to propose reforms in European legislation only. The global nature of the issues involved, however, suggests that a reduction of the software copyright scope of protection should not be sought only in the EU, but rather worldwide.

4.3. Patent Protection to Limit the Scope of Copyright

The second point this article makes relates to the role of patents for software-related inventions. Specifically, the article argues that maintaining patent protection for computer-related inventions is necessary in order to keep up the distinction between copyrightable and patentable objects in the field. The highly debated issue of software as patentable subject matter, both under the European 'as such' exclusion and under the recently formulated American Bilski test[111] is, thus, clearly missing the real point.

As long as the examination process is properly done, patent law should not raise the same concerns as copyright, and inventors should be rewarded only for novel, unobvious inventions. It should be noted that no idea/expression dichotomy exists in patent law. If an idea meets the patentability requirements, it will be awarded patent protection without covering the expression of ideas.[112] However, the bigger risk of overprotection might arise if patents are granted too freely and interpreted too broadly by the patent officers and courts.[113]

It is evident that the current patent system includes various aspects that urgently need to be changed in order to assure the promotion of progress in the software field. In Europe, the difficulties resulting from the 'as such' exclusion of computer programmes from patentable subject matter is well-known. Additionally, the abstractness of software patent claims, as well as the general lack of relevant prior art and the insufficient level of

111 *In Re* Bilski, F.3d, 2008 WL 4757110, 88 U.S.P.Q.2d (BNA) 1385 (Fed. Cir. Oct. 30, 2008).

112 W Cornish, and D Llewelyn, *Intellectual Property: Patents, Copyright, Trade Marks & Allied Rights* (5th ed, 2003), at 173-207.

113 R M Ballardini, "The Software Patent Thicket: A Question of Disclosure", (2009) 6:2 SCRIPT-ed 207, http://www.law.ed.ac.uk/ahrc/script-ed/vol6-2/ballardini.asp (accessed 16 Nov 2009).

disclosure in the software patent applications, have led to the issuing of overly-broad, not novel and obvious patents in the field. Overall, this has degenerated in recent years into an inefficient system that has been the source of harsh criticism, particularly in the United States.[114]

Finding solutions for improving this rather imperfect, inefficient system is a priority. Eliminating computer programmes from the patentable subject matters or excessively reducing the possibility of getting patents on software, however, not only does not solve the problem, but also increases the danger of expanding copyright on programmes aggravating both risks of software copyright over- and under-protection.

On this basis, this paper suggests some refinements to patent law in order for it to better accommodate the needs of software. Focus is placed on European patent regulations.

4.3.1. The Role of Patents for Computer-Related Inventions

Software-related patents, as they currently stand, possess some undoubtedly negative aspects. Notwithstanding these problems, there are some important factors that should be considered when discussing policies on patent protection for computer-related inventions.

A first crucial point refers to the fact that the availability of patents for software-related inventions helps to reduce some of the possible concerns of both the risks of over- and under- protection involved in copyright and software functionality.

On one hand, by affording legal protection to the functional elements of software, patents fill one of the deepest gaps in copyright law. Evidently, the role of patents would not be degraded by limiting copyright protection only to literal copying, as this article has proposed.

On the other hand, there is reason to believe that the granting of patent protection might be the only feasible means by which to maintain a sufficiently clear distinction between copyrightable and patentable objects in software. This is true especially as far as non-literal copying of the code is kept under the umbrella of copyright law.[115]

These assumptions become more realistic when assessed in view of the American experience. In the US, the past has shown that limiting patent protection by categorically excluding computer programmes not only risks leaving the most valuable parts of software (i.e. its functions) unprotected, but might also lead to the expansion of the scope of copyright.[116] As seen before, the initial rejection of software patentability, for instance, led to cases such as *Whelan*, where copyright had been widely extended to protect elements beyond the simple literal expression of the programme's executable code. The availability of patent law, instead, has rendered broad copyright protection for the structural and functional elements of software less important.[117] For example,

---

114 See, for instance, Federal Trade Commission, *To Promote Innovation: the Proper Balance of Competition and Patent Law and Policy. A Report by the Federal Trade Commission*, October 2003, Chapter 3-V.

115 D Karjala, "Distinguishing Patents and Copyright Subject Matter", (2003) 35 *Conn. L. Rev.* 2, at 439-524.

116 *Ibid*.

117 See note 39 above.

empirical analyses[118] have demonstrated that *Lotus* has had a considerable impact on patenting, as the number of patent applications filed appears to have increased more dramatically for interface firms than it has for others.

This trend is evidenced also on the European front. The potential 'thick' protection allowed under the software copyright Directive appears to be a remedy for filling the gap left by the limited availability of patent protection for computer-related inventions.

Therefore, it seems reasonable to argue that if the scope of software-related patents is reduced too much courts might be likely to try to correct with copyright law what they consider an illegitimate appropriation of the fruits of someone else's creativity.[119] As extensively discussed earlier, however, this might not only contravene basic copyright principles, but might also be highly detrimental for overall software innovation.

This analysis suggests that it is vital to retain patent protection for computer-related inventions (at least to a certain extent). On the contrary, the clamour over the exclusion of software from patentable subject matter, which is often at the centre of discussion when addressing questions of software IP protection,[120] is missing the core point. In the same way as copyright, thus, in respect to patent protection the relevant question lies in the scope to accord to software-related patents. The next section briefly discusses how such a goal could be reached in Europe.

4.3.2. Redefining European Software Patents – Some Remarks

The most controversial issue in patenting computer-related inventions in Europe lies in the interpretation of the 'as such' exclusion.

Article 52(2)(3) EPC excludes computer programmes 'as such' from patent law. Accordingly, the EPO accepts only computer-implemented inventions (CII)[121] as patentable subject matter. The interpretation of this exclusion with respect to computer programmes has been anything but simple. Specifically, the difficulty of limiting the types of admitted claims relating to computer programmes is apparent both from the case law of the EPO Boards of Appeal[122] and from many national rulings.[123]

The interpretation of the 'as such' exclusion revolves around the 'technical' criterion, i.e. to be patentable subject matter inventions must be 'technical' in nature. The difficulty of applying the technical requirement to software, however, has brought the EPO to

---

118 J Lerner, and F Zhu, "What Is the Impact of Software Patents Shifts? Evidence Form *Lotus v. Borland*", (2007) 25 *International Journal of Industrial Organization* 3, at 511-529.

119 See also note 115 above.

120 See, for example, Court of Appeal (England and Wales), *Aerotel Ltd v Telco* [2006] EWCA Civ 1371; *In re* Bilski , F.3d, 2008 WL 4757110, 88 U.S.P.Q.2d (BNA) 1385 (Fed. Cir. Oct. 30, 2008).

121 Computer-Implemented Inventions are "Inventions whose implementation involves the use of a computer, computer network, or other programmable apparatus, the invention having one or more features which are realised wholly or partly by means of computer programs". See "Patents for Software? European Law and Practice", European Patent Office (2008).

122 See, for example, T 0931/195 *Controlling Pension Benefit System/PBS Partnership* [2001] OJEPO 441; T 0258/03 *Auction Method/Hitachi* [2004] OJEPO 575; and T 0424/03 *Microsoft/Data transfer expanded clipboard formats* [2006]. See also the critic in Court of Appeal (England and Wales), *Aerotel Ltd v Telco* [2006] EWCA Civ 1371.

123 See Aerotel/Macrossan note 120 above. For a general discussion see 57 above.

embrace many different and inconsistent approaches, overall leading to a general lack of legal certainty in the field.[124]

It is not within the scope of this article to deeply analyse the issues and case law related to the interpretation of the technical requirement in CII. The author has discussed the matter elsewhere.[125]

However, it is important to direct attention to the need to develop uniform and precise rules to clarify the conditions for allowing patentability of CII under European patent law. Computer software penetrates every aspect and almost all fields of technology of our society. The question of patent protection for software-related inventions, thus, becomes a question of applying a system of protection to one of the fastest-growing economic sectors.

4.3.2.1. The Dilemma Ought to Be Resolved

EPO jurisprudence has demonstrated how assessing the 'as such' exclusion under the concept of 'technical invention' creates confusion and uncertainty. The main difficulty was not to identify what elements of CII were technical and what were not, but rather to be able to separate the two sets.

To ease the assessment of 'technical invention' in the software filed, in the 2000s, the EPO started to admit only computer implementations to 'invention' status. Under such an approach, named the 'any hardware approach', software-related inventions involving the use of some sort of hardware (e.g. a general computer), are considered 'technical'. The technicality of the invention is checked only while assessing its inventiveness.

Although it appears that under the 'any hardware approach' the seemingly absolutist 'as such' exclusion of computer programmes has lost any real meaning, this is undoubtedly an easier way of interpreting European patent law without breaking the EPC principles. In this respect, the EPO's switch from 'invention' to 'inventiveness' represents an important step towards an innovative, modern interpretation of patent law for computer programmes. On the other hand, being the problem-and-solution-approach used to assess inventiveness made on a predominantly technical basis, it leaves the 'technical/non-technical' dilemma unsolved.

Under European patent law, in fact, to be inventive, an invention must provide 'a technical solution to a technical problem' (the so-called 'problem-and-solution' approach).[126] Features which, either independently or in combination with other features, do not make any contribution to the technical character of an invention are not taken into consideration for assessing the inventive step.[127] This might occur when a feature only contributes to the solution of a non-technical problem, such as a problem in one of the excluded fields like computer programmes or business methods 'as such'.

The EPO Enlarged Board of Appeal has recently been called upon to provide some clarification over the interpretation of the excluded subject matter, specifically in the

124 See note 57 above.

125 *Ibid*.

126 See EPO Guidelines for Examination (2007).

127 See T 641/00 Two Identities/COMVIK [2002] OJ 7/2003, 352.

field of computer programmes.[128] This is an excellent opportunity for the EPO to finally shed some light on this highly controversial area of patent law. The present article suggests that the 'as such' exclusion in computer programmes should be interpreted as follows.

The article supports the interpretative line derived from the 'any hardware approach', as it comes to the interpretation of 'invention'. Accordingly, programmes implemented in hardware should always qualify as 'technical inventions'; the focus should then be placed on 'inventiveness'. The assessment of the inventive step, however, needs some refinement. Specifically, assessing the inventive step after considering all the features of the invention (i.e. technical and non-technical) is essential in order to promote a more coherent and transparent practice.

Such a system would, on one hand, eliminate the impossible task of having to separate technical features from non-technical ones, allowing examiners to concentrate their efforts on deciding whether the invention is actually inventive with respect to the current state of the art. On the other, claimed non-technical features might well contribute to the aggregate technical character when viewed as a whole, i.e. combined with all other claimed features.

The question of whether a technical effect exists arises when it is unclear whether the claimed feature actually describes something physical or its behaviour, or, instead, something abstract. In this regard, the EPO case law has held that as long as the claim 'as a whole' is directed to a functional system there should be sufficient technical effect in the real world to avoid Article 52's exclusion. The EPO case law, in fact, clearly shows that the relevant question is whether the claim 'as a whole' identifies a technical solution to a technical problem.[129]

The fact that the technical character is to be sought from the claimed features 'as a whole', suggests that a technical effect should exist for at least one, *but not necessarily all*, of the claimed features. To evaluate the overall technicality of the invention at hand, thus, both technical and non-technical features should be considered.

Solving the technical requirement issue would certainly represent a remarkable step towards a more efficient, clearer system in Europe.

4.3.2.2. Other Reasons of Concern

Applying the 'technicality' principle to CII is undoubtedly the main problem of the European patent system. It should be mentioned, however, that other concerns also surround the software patent environment. In particular, the increase of patents filed within a short period, the abstractness and unclearness of the patent claims, the lack of relevant prior art repositories, as well as the limited disclosure requirements, have recently led to anxiety over the 'quality' of the patents granted.[130]

---

128 See Case G3/08: referral under Art. 112(1) b) EPC by the President of the EPO (Patentability of programs for computers) to the Enlarged Board of Appeal pending under Ref. N° G3/08 (23.12.2008).

129 See, for instance, T 0258/03 Auction Method/Hitachi [2004] OJEPO 575.

130 See, for instance, note 114 above.

This imperfect system evidently calls for an immediate and quick action plan to improve and better support the patent examination process.

Efforts should, therefore, be directed also to promoting an efficient patent system that maintains an appropriate level of novelty, inventiveness, as well as disclosure in the patents it issues. Policy changes could include, for instance, the promotion of enhanced disclosure under the sufficiency of disclosure requirement, so as to reduce the abstractness of the software patent claims and the consequent difficulty of defining the boundary between allegedly new, innovative inventions and prior art.[131] Enhanced disclosure might discourage the filing of very uncertain applications and thus reduce also the number of patents filed. Additionally, even though various projects have already been launched in order to improve the prior art repositories,[132] a lot of work still needs to be done in this area. In particular, both private and governmental projects to clear the registries should be supported.

What is certain is that computer-related inventions need to be kept under the scope of patent protection. This is the only workable means for the overall IP ecosystem (in particular patent and copyright law) to properly function in the software context.

5. CONCLUSIONS

That copyright law fails in providing an adequate level of protection to computer programmes is a widely recognised problem. That patent law should be applied differently in order to effectively serve the purpose of enhancing innovation in the software field, is constantly under discussion. There is no reason for the current system to remain in place if it is causing problems of over- and under- protection and is potentially stifling innovation in the field.

By proposing some refinements to both bodies of European law, the present article has put forward a workable model for a more transparent system, which is consistent with both the copyright and patent law principles.

Specifically, on the copyright front, this article has advocated the adoption of explicit statutory limitations against non-literal infringement to keep the scope of software copyright protection within the sphere of literary works of art. This should be complemented by the ready availability of patents to protect the functional aspects of the programme, as well as DRM measures, used in combination with trade secrecy and contracts. In particular, keeping software within the umbrella of patent law is essential in order to impede copyright from expanding to protect the utilitarian elements of software, which would allow it to act in a 'patent-like' manner. The highly controversial question of software as patentable subject matter, both under the European 'as such' exclusion and under the recently formulated American Bilski test is, thus, missing the real point. Efforts should, instead, been concentrated in the amelioration of the currently existing patent rules, in order to assure a proper level of novelty, inventiveness and sufficient disclosure in the patents issued.

---

131 See *EPC*, Article 83.

132 See, for instance, Patent Commons Project: http://www.patentcommons.org/ (accessed 16 Nov 2009); Prior aRT and Software Patents project: http://wiki.mozilla.org/Legal:Prior_Art (accessed 16 Nov 2009); Open Source As Prior Art (OSAPA) project: https://www.linux-foundation.org/en/Osapa (accessed 16 Nov 2009); PatExpert: http://www.patexpert.org (accessed 16 Nov 2009).

ESSAY II

Rosa Maria Ballardini

Software Patents in Europe. The Technical Requirement Dilemma

Journal of Intellectual Property Law & Practice 2008(3)9:563-575

Peer reviewed

Software patents in Europe: the technical requirement dilemma

Rosa Maria Ballardini*

Time for radical change

Patents are an efficient instrument for protecting innovation by encouraging technological progress. However, they work differently depending on the industry to which they are applied. When new technologies and thus new types of inventions arise (as in the case of software), moulding the system to meet the features of the new developments is a challenge. Both the newness and the rapid development of software engineering have brought the patent system to its knees, distorting the general patent environment.

This article assesses the problems involved in applying patent law to computer programs. Its perspective arises from the specific features of computer programs, their structure, and way of functioning. Its main focus is on the 'technicality' requirement involved in evaluating and granting patents for computer-related inventions in the European tradition. A comparative analysis between the approaches followed in the EPO and at the national level will illuminate the current situation of legal uncertainty and incoherency. The study concludes that the traditional European system, where only 'technical' inventions receive patent status, has become obsolete under current rules. As technology advances, the law must adapt to face the new needs of the society to which it applies. A radical change is, therefore, a current priority.

Key issues

- The practice to be followed regarding computer program patentability in Europe has been long discussed, but significant differences remain between the EPO approach and national European jurisprudence.
- Extensive case law in the field shows that the 'technical requirement', main pillar of the traditional European patent system, as applied to computer programs, has repeatedly proven inappropriate and confusing.
- A harmonized system, focusing mainly on the inventive step and with a relatively narrow patent scope would provide a reliable basis for software patent protection, eliminating the contradictions that currently exist with the application of the technical criterion.

Patenting software in Europe

Definition of 'software'

What exactly do we mean by 'software'? The significance of words such as 'computer program', 'algorithm', 'source code', and 'object code' is not needed in everyday software engineering.[1] However, when they become part of legislation, the need for a precise definition of the terms arises.

Although no European legislation defines 'software', computer science and software engineering have formulated hundreds of quasi-official definitions. For example, the IEEE Standard Glossary of Software Engineering Terminology[2] states that software includes:

> Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

It follows that 'computer program' is just a part of software and that patenting a computer program is not the same as patenting software. It is commonly understood that software includes both computer program and its ancillary materials (eg documentation, computer file, or data). However, these terms can be, and often are, used interchangeably, on the assumption that the

* PhD Candidate at HANKEN-Swedish School of Economics and Business Administration; member of the INNOCENT Graduate School in IP law, IPR University Center, University of Helsinki. The author thanks Professor Niklas Bruun, Associate Professor Marcus Norrgård, and Dr Mikko Välimäki for the valuable comments, although the usual disclaimer applies. Sincere thanks also go to the CIMO organization for financially supporting this research.

1 R Sarvas, 'A software engineering view to the decisions of the European Patent Office concerning computer programs', Helsinki University of Technology, 2001, pp. 11–16.

2 IEEE Std 610.12-1990, *Standard Glossary of Software Engineering Terminology* (The Institute of Electrical and Electronics Engineers, New York, 1990).

difference between the two is clear. In this paper, the same meaning is given to both the term computer program and software in the context of patent protection.

EPC requirements and its interpretation

In the European system, the definition of patentable subject matter is enshrined in the European Patent Convention (EPC) 1973[3] and subsequent modifications, as ratified into the national legislations of all the EPC Member States.

Article 52 states what is patentable. Paragraph 1 sets the four patentability requirements: invention, novelty, inventive step, and industrial application. Although the requirements of novelty, inventive step, and industrial application are defined in the convention, 'invention' is not, it being inferred in practice that an invention must be 'technical'.[4] Paragraphs 2 and 3 specifically refer to the patentability of software, excluding computer programs 'as such' from the category of 'inventions' and thus from patentability. Thus, computer programs 'as such' are not considered to be technical under European patent law.[5]

The EPC Implementing Regulations[6] and the EPO Guidelines for examination[7] also play a fundamental role in the application and interpretation of the EPC. However, neither the EPC nor the Implementing Regulations or Guidelines provide any rational justification for the exclusion of computer programs from patentability, except that they are considered as 'non-technical'.[8] Therefore, the key problem lies in deciding when and under what conditions an invention relates to technical features and functions.

Reasons for the exclusion

The drafting of the EPC took more than a decade during the 1960s and 1970s. Discussions on exclusions from patentability coloured the debate during those years. However, it was not until the second round of negotiations that the proposal to include computer programs within the category of unpatentable subject matters was put forward by some non-governmental organizations.[9]

The decision to exclude computer programs, which resulted from a bitter debate, was strongly opposed by the UK. A principal argument against exclusion was that, in a field as uncertain as computer programs, one should be flexible in the development of precedents.[10] Barring computer programs 'as such' from patentability and, in so doing, leaving them to be protected by 'indirect means' were the antithesis of that process.[11]

Furthermore, as the first version of the EPO Guidelines shows, at that time computer software was considered an independent component, easily separable from the hardware itself, which could thus be excluded from patentability per se. Due to the technological progress and evolution, however, the nature of software has radically changed. Although it remains clear that computer software per se is not patentable, drawing the boundary between prohibited and admitted subject matter has become a challenging task.[12]

What is 'technical' in software?

Patentable inventions must be technical in nature. The description and identification of the technical nature and effect of computer programs does not seem to present much trouble in theory. For example, technical elements might be found in the apparatus or system (that is, the computer containing the novel software), the process which the program performs, or even in the storage medium or signal carrying the program.[13] This identification, however, becomes less trivial in practice, mainly due to the intangibility of computer software given its dual nature of being simultaneously a 'writing' and possessing 'behaviour'.[14] The problem stands out immediately once we compare traditional inventions, which are closer to those for which the

3 Convention on the grant of European patents of 5 October 1973 (European Patent Convention).

4 K Beresford, *Patenting Software Under the European Patent Convention* (2000) 28–45.

5 IPR Helpdesk, Patentability of Computer Programs, 2005, http://www.ipr-helpdesk.org

6 Rules 27 and 29 are the most relevant with concern to software patentability.

7 EPO, *Guidelines for Examination*, Part C, Chapter 4 (2007).

8 R Sarvas, 'More Ambiguity in European Software Patenting', Helsinki Institute for Information Technology (HIIT), 2001, http://www-old.hiit.fi/de/core/MoreAmbiguityInEuropeanSWpatenting.Pdf

9 J Pila, 'Article 52(2) of the Convention on the Grant of European Patents: What did the framers intend? A study of the travaux préparatoires', 36 IIC 755 (2005). See also Beresford, above n 5, 12–22.

10 See H MacQueen, G Laurie, C Waelde, and A Brown, *Contemporary Intellectual Property: Law and Policy* (1st edn, 2007) 508–535.

11 ibid.

12 I Lloyd, 'Software patents after Fujitsu. New directions or (another) missed opportunity?', Case Commentary 1997 (2), *The Journal of Information Law & Technology (JILT)*.

13 Beresford, loc cit.

14 P Samuelson, R Davis, MD Kapor, and JH Reichman, 'A manifesto concerning the legal protection of computer programs', *Columbia Law Review* (1994), section 94. see also DL Burk and MA Lemley, 'Designing Optimal Software Patents' in R Hahn (ed.), *Intellectual Property Rights in Frontier Industries: Software and Biotechnology* (2005). Available at SSRN: http://ssrn.com/abstract=692044.

patent system was initially designed, with computer-related inventions. A traditional invention usually has physical implementations with concrete, tangible components and with a functionality that is often easily understood even by someone not skilled in the art.[15] In contrast, a computer-related invention rarely possesses any geometrical representation, since its components have no physical implementation, and its result is intangible.[16] It is this far harder to evaluate the concrete and 'technical' applicability of a computer-related invention, and more interpretation is required in order to assess its patentability.[17]

Case law background of the EPO approach

The exclusion of computer programs under Article 52 EPC does not mean that such patents are rarely applied for or granted. On the contrary, more than 6% of European patent applications[18] lie within the area of data processing[19], and hundreds of patents on software are granted each year both by the EPO and by national offices. Recent statistics show that roughly 20–30,000 patents for computer-implemented inventions were issued by the EPO by 2004.[20]

The EPO Technical Boards of Appeal have had the opportunity to clarify how a computer program 'as such' should be characterized.[21] Over the past 20 years, the Boards have issued a series of important decisions aimed at defining the approach to be followed. However, the precise definition of 'technical contribution or effect' has remained elusive and the criterion of technicality has often led to arbitrary and contradictory decisions.[22]

The analysis of some of the major judgments of the Technical Boards of Appeal concerning the 'technical' requirement indicates three different approaches: the 'technical contribution' approach, the 'technical effect' approach, and the 'any hardware' approach.[23] These practices have themselves been interpreted in different ways, generating slight divergences between the reasoning in each decision.

The 'technical contribution' approach

In July 1986, the EPO Technical Board of Appeal delivered a landmark decision in *Vicom*.[24] The claims were to both a method of digitally processing images and an apparatus for carrying out the method. The Board held that the claims were not to a computer program 'as such'.

In its reasoning, the Board introduced a new distinction between 'pure' mathematical algorithms and 'applied' algorithms to be used in a process, seeking to distinguish between an abstract concept and technical signals.[25] It concluded that a process cannot be excluded from patentability for the sole reason of being based on an algorithm. Instead, even if made of non-patentable elements, it might be considered as both an invention and patentable as long as a 'technical contribution' is made to the known arts.[26]

The 'technical contribution' approach was also applied in later EPO cases (eg *Text Processing/IBM*[27] and *Koch and Sterzel*[28]), as confirmation that a computer implementation is not capable of describing an 'invention' if the contribution provided resides solely in a computer program.[29]

Under this approach, the examiner should find whether there was an inventive step in order to ascertain the existence of an invention. Under the EPC rules access to patentability requires first an invention and, only then, need novelty, inventive step, and industrial applicability must be checked. This being so, the legitimacy of the contribution approach seems somehow lacking. Moreover, vagueness remained as to the type of invention considered as making the right technical contribution to fall within the scope of patentable subject matter.

These reasons led during the 1990s to the overcoming of the contribution approach by another interpretative line, requiring a 'technical effect'.

15 Sarvas, n 1.

16 ibid.

17 ibid.

18 European Patent Office, Annual Report 2004, at 20 *et seq.*, IPC- Class G 06.

19 There are also other fields with computer program-related applications, such as electrical engineering, electronic, measuring and regulation systems as well as information technologies. Exact statistics are, therefore, rarely possible.

20 See also European Software Patent Statistics, http://eupat.ffii.org/patents/stats/index.en.html.

21 AL Guadamuz, The software patent debate. *Journal of Intellectual Property Law & Practice*. 2006; 1: 196–206. Available at SSRN: http://ssrn.com/abstract=886905.

22 ibid.

23 See classification made in the Court of Appeal (England and Wales), *Aerotel Ltd v Telco* [2006] EWCA Civ 1371.

24 T 0208/84 *Computer related invention/Vicom* [1987] OJEPO 14.

25 P Leith, 'Patenting Programs as Machines' (2007) 4:2 SCRIPT-ed 214, http://www.law.ed.ac.uk/ahrc/script-ed/vol4-2/leith.asp

26 C Laub, Software patenting: legal standards in Europe and the US in view of strategic limitations of the IP systems. *The Journal of World Intellectual Property*. 2006; 9: 344–372.

27 T 0038/86 *Text processing/IBM* [1989] OJEPO 118.

28 T 0026/86 *X-Ray Apparatus/Koch & Sterzel* [1987] OJEPO 585.

29 GI Zekos, Software Patenting. *The Journal of World Intellectual Property*. 2006; 9: 426–444.

The 'technical effect' approach

The EPO abandoned the technical contribution approach in the 1990s, according to the EPO Guidelines 2001. Meanwhile in 1999, in two important decisions involving IBM,[30] the Technical Boards of Appeal started elaborating a slightly different reasoning, later termed the 'technical effect' approach.

These clearly confirmed that computer programs were to be considered patentable once they had a technical character.[31] The mere interaction between a program and a machine—the internal electrical changes to the computer induced through the execution of a program by hardware—was insufficient to endow it with technical effect.[32] However, a technical character could be found

> in the further effect deriving from the execution (by the hardware) of the instructions given by the computer program. Where said further effects have a technical character or where they cause the software to solve a technical problem, an invention which brings about such an effect may be considered an invention which can, in principle, be the subject-matter of a patent.[33]

In other words, it is possible to claim a computer program on its own or on a carrier, provided that such a program causes a further technical effect, when being carried out on a computer.[34]

The 'any hardware' approach

During the past 7 years, the practice of the EPO Boards of Appeal has shifted again. More new case law emerged, starting with an important decision taken in *Pension Benefit* in 2000.[35] These decisions show that, in order to decide whether a computer program has technical character, the examiners must ask 'whether the claim involves the use of or is to a piece of physical hardware, however mundane. If it is, Art. 52(2) does not apply'.[36] It seems, therefore, that under the 'any hardware' approach, it is easy enough to satisfy the 'invention = technical' test, it being sufficient to refer to some features of the hardware in the patent claim.[37] The technical nature of the invention, instead, is established here, when assessing the inventive step: if a non-obvious *technical* solution is provided to solve a technical problem, the invention cannot be considered as inventive.[38] Under this approach, the examiners must first investigate whether there is an invention and only then consider whether that invention makes a technical contribution to the state of the art. The first impression is, therefore, that this practice is more legitimate than the previous 'technical contribution' one. However, under the 'any hardware' approach, any computer implementation alone can be considered an 'invention' by the office.

The trilogy

The 'any hardware' approach of *Pension Benefit* was extended (although in contradictory ways) to other cases, of which *Hitachi* (2004)[39] and *Microsoft/Data transfer expanded clipboard formats* (2006)[40] assume major relevance.

Pension Benefit is important since it clarifies the EPO position on business methods patentability, another excluded category under Article 52(2) EPC. More often than not, since business method cases overlap with the computer program exclusion, the same observations apply.

The claims were to both a method of controlling a pension benefit system and an apparatus for performing the method (ie, a computer suitably programmed). With reference to the method claim, the Board found no technical effect:

> where a claim is to a method which consists of an excluded category, it is excluded by Article 52(2) even if hardware is used to carry out the method.

Thus, the method claim was caught by Article 52(2).[41] The apparatus claim was then analysed and it was held that 'a computer programmed to carry out an unpatentable method did not fall under Art. 52(2)'. The fact that a 'concrete', physical thing was involved was enough to place it out of the reach of excluded subject matter. However, the apparatus was considered lacking inventiveness as the method (unpatentable)

30 T1173/97 *Computer Programs Product/IBM* [1999] OJEPO 609; T 0935/97 *Computer Program Product/IBM* [1999] OJEPO 609.

31 Newman, n 10.

32 Below n 33.

33 ibid.

34 A Soininen, 'The Software and Business Method Patent Ecosystem: Academic, Political, Legal and Business Developments in the US and Europe' in A Soininen (ed.), *Patents in the Information and Communications Technology Sector—Development Trends, Problem Areas and Pressures for Change* (Lappeenranta University of Technology, 2007) 44–55.

35 T 0931/195 *Controlling Pension Benefit System/PBS Partnership* [2001] OJEPO 441.

36 As summarized in *Aerotel Ltd v Telco*, n 26.

37 Laub, n 29.

38 Soininen, loc cit.

39 T 0258/03 *Auction Method/Hitachi* [2004] OJEPO 575.

40 T 0424/03 *Microsoft/Data transfer expanded clipboard formats* [2006].

41 See W Cook and G Lees, 'Test clarified for UK software and business method patents: but what about the EPO?' Comments: [2007] EIPR 115–118.

carried out was obvious to the skilled men: same result, different reasoning.[42]

In *Hitachi*, the claim concerned an automatic auction method executed in a server computer, and the apparatus for carrying that method out. The Board first examined the apparatus claim and, following the same reasoning as in *Pension Benefits*, stated that the computer programmed to conduct a business method (ie an unpatentable method) was not excluded by Article 52 since it included clearly technical features 'such as a server computer, client computers and a network'. With respect to the method claim, the Board, disagreeing with *Pension Benefits*, continued that it was an invention, patentable for the same reason that it implied technical features throughout the hardware.[43] However, when assessing inventive step, the Board found that no non-obvious technical solution had been provided. Thus, both apparatus and method claims were found to be invalid.[44] Again: same result, but different reasoning.

Finally, *Microsoft* contradicts both *Pension Benefit* and *Hitachi*, producing opposite contrasting outcome. Here the patent claim was directed to a computer-readable medium having computer-executable instructions (namely, a computer program) on it to cause the computer system to perform the claimed method. Following the example in *Hitachi*, the Board considered both method and apparatus claims patentable. However, when examining inventive step, the Board did not treat the unpatentable computer program 'as such' as part of the prior art as in *Pension Benefit* and *Hitachi*. Instead, the invention was considered new and non-obvious on conventional grounds (although no reasoning was given).[45]

The contradiction mainly lies in the Board here analysing both method and apparatus claims, without excluding the contribution of the unpatentable subject matter. This reasoning clearly departs from that of *Pension Benefit* and *Hitachi*, where the excluded subject matter (residing in the method claim), although eligible for 'nvention' purposes, was later blocked in the assessment of the inventive step. The *Microsoft* approach cleverly gets around, without breaking, the 'as such' exclusion of Article 52 EPC and, in so doing, opens the way to the patentability in principle of any computer program in Europe. What, if anything, still remains of the exclusion of computer programs from European patent law seems, therefore, to be questionable under current rules.

42 See also V Salomon, Patenting computer software and business methods in the UK. *Communications Law.* 2007; 12.

43 ibid.

44 See Cook & Lees, n 41.

Summary

The EPO Technical Boards of Appeal have long sought to define the relevant criteria for the patentability of computer programs. To this end, the main focus has been on the question of whether there is an invention and, consequently, whether such an invention is technical in nature and makes a technical contribution. Thus, the scope of patent protection has been narrowed to the requirement of technicality. However, the difficulty in pinpointing such a criterion in the software sector has caused the Boards to embrace various and inconsistent approaches, leading to a general lack of legal coherency in the field. This has been highlighted in particular in the most recent 'any hardware' approach, where the same situation led to different results.

Applicants thus lack the chance to predict the results of their investments with reasonable certainty or to decide whether it is worth investing resources in borderline inventions, for example when a mix of technical and commercial features is present. Attempts to clarify whether the subject matter has the right sort of technical character to bring it within the meaning of invention have repeatedly failed, raising the question whether we should keep on relying on the 'technical requirement' or modify the current rules and try to look for some more workable solutions.

National approaches

Despite various attempts to harmonize patent legislation in Europe, patent protection is still largely granted at national level. Though national legislation is usually in line with international treaties such as the EPC[46] and TRIPs,[47] notable differences still reside in the interpretation of such laws and in the practices adopted by national patent offices and courts. These divergences are particularly accentuated in the sector of computer programs.

In this part, the practices and policies adopted by some of the EPC major players—the UK, Germany, and France—are discussed. Since the UK and Germany are where most patent litigation on software takes place, they represent a significant sample of the overall European trend. The main focus is posed on the interpretation given by the national courts to the 'technical requirement' as the main pillar of the applicability of European patent law. This analysis highlights the

45 ibid.

46 Above n 4.

47 WIPO Treaty on trade related aspects of intellectual property rights, Morocco, 15 April 1994 (TRIPs).

unharmonized situation in Europe, throwing light on the general legal uncertainty.

United Kingdom

The Patent Act 1977 section 1(2) implements Articles 52(2) and (3) EPC, and explicitly excludes programs for computers (among other things) from patentability. The UK-IPO also produces guidelines[48] (constantly updated) for the general interpretation and application of patent law, with a specific section on computer program patentability.

The UK courts have been reluctant to afford software patent protection and, with few exceptions,[49] the majority of applications have been rejected under section 1(2).[50] This has created a deep rift between EPO practice and the UK-IPO approach, raising the argument that software applications have been treated in an excessively harsh and restrictive way in the UK.

Following the 'technical contribution' approach

The first important judgment on software patentability was the Court of Appeal decision in *Merrill Lynch* in 1989.[51] By the time this case reached the Court of Appeal, *Vicom* had already been published by the EPO and the 'technical contribution' approach applied. Following the same reasoning as *Vicom* the hearing officer argued that, in order to judge the patentability of matters excluded under section 1(2), it was necessary to assess the technicality of the contribution on which the invention was an improvement.[52] Following this reasoning, the invention claimed in *Merrill Lynch* was considered unpatentable on the ground that it was no more than a method of doing business.

The British courts used the 'technical contribution' test for more than a decade, including *Gale*[53] and *Fujitsu*.[54] However, those cases reflected the difficulty experienced by the judges in understanding what was to be considered 'technical' in the added contribution.[55]

Reference to inventive step, the non-obviousness of the invention, was made in neither *Vicom* nor *Merrill Lynch*. However, while the jurisprudence of the EPO evolved to focus increasingly on whether a non-obvious technical solution is provided to a technical problem, the UK-IPO adhered to a regime based on the technical contribution an invention provides, rather than on its inventiveness. This resulted in some major differences between the UK-IPO and EPO practices during the 7 years between *Merrill Lynch* and *CFPH*,[56] considered below.

The 'two-step' approach

In 2005, three important High Court judgments embraced a different approach, clearly departing from the line followed thus far and steering the UK for a while along the same track as the EPO.[57]

The first two decisions—*Shoppalotto.com*[58] and *Halliburton v Smith*[59]—departed slightly from the *Merrill Lynch* reasoning without radically abandoning it. Their main thrust was that, in order to be an invention, the contribution should have relied on a 'technical effect', and not just on excluded subject matter.[60]

Finally, in *CFPH*,[61] the vagueness of the *Vicom* 'technical contribution' approach caused the judge to formulate a different test comprising two steps that, in a way, resemble the then practice of the EPO:

(a) Identify what it is, in the advance in the art, that is said to be new and non-obvious (and susceptible of industrial application);

(b) Determine whether the advance in the art is both new and not obvious (and susceptible of industrial application) under the description 'an invention' (in the sense of Article 52 EPC).[62]

The focus was mainly on the assessment of novelty and non-obviousness (inventive step), rather than on the technical nature of the invention. This new test temporarily announced the death of the 'technical contribution' approach in the UK.[63]

It brought the UK approach closer to that of the EPO. However, it strongly criticizes the EPO's reliance on a vague definition of 'technical' when paraphrasing the exclusions of Article 52. A few months later, an appeal

48 Manual of Patent Practice guidance for interpreting the Patent Act 1977, http://www.ipo.gov.uk/patent/p-decisionmaking/p-law/p-law-manual/p-law-manual-practice/p-law-manual-practice-patent1977.htm

49 An example being *Gale* [1991] RPC 305, reversed by the Court of Appeal.

50 Lloyd, n 14.

51 [1989] RPC 561.

52 See Cook, n 44.

53 [1991] RPC 608.

54 [1997] EWCA Civ 1174.

55 This uncertainty has been clearly stressed by Nicholls LJ when evaluating Gale's *Application*: 'I confess to having difficulty in identifying clearly the boundary line between what is and what is not a technical problem for this purpose'.

56 See Lloyd, n 14.

57 ibid.

58 [2005] EWHC 2416.

59 [2005] EWHC 1623.

60 ibid.

61 [2005] EWHC 1589.

62 D Bainbridge, 'Court of Appeal parts company with the EPO on software patents', Computer Law and Security Report 23 (2007) 199–204.

63 ibid.

judgment in *Aerotel v Telco* and *Macrossan*, was again to move UK practice further from that of the EPO.

Towards a new era

The *Aerotel/Macrossan* judgment[64] represents the most important recent UK decision in the field of computer program patentability. The particular impact this judgment has had worldwide, together with its effect on the entire British jurisprudence, makes this case worthy of special consideration.

Aerotel/Macrossan, a judgment of the Court of Appeal of England and Wales in October 2006, relates to two different appeals from the Patents Court. The first (Macrossan) involved a patent rejected on the grounds of being both a method of doing business and a computer program as such. The second concerned a patent granted to Aerotel and its infringement action against Telco and others. In this case, the appeal was granted and the patent application considered more than a business method as such: it was a new 'physical combination' of hardware and thus an invention.

Probably the most important contribution this decision made is that the presiding appellate judge, Lord Justice Jacob, having openly refused to follow the EPO approach in interpreting the exclusions of Article 52 EPC, suggested a new 'four-step' test that retroactively catapulted the UK to the *Merrill Lynch* and *Vicom* way of thinking, negating the position in *CFPH*. The four steps of the test are:

1. Properly construe the claim.
2. Identify the actual contribution in the light of the prior art.
3. Ask whether the actual contribution falls solely within the excluded subject matter.
4. Check whether the actual or alleged contribution is actually technical in nature.[65]

The fourth step, that is to assess the 'technical contribution' of the invention, may be unnecessary. There is reason to believe that Jacob LJ would have even preferred to omit the last step, but felt bound by the precedent in *Merrill Lynch*.[66]

The 'four-step' test: consequences

The *Aerotel/Macrossan* decision rises some points of concern because, under the 'four-step' test, the UK propensity for patenting software is much more restrictive than in the rest of Europe. Further, the *Aerotel/Macrossan* doctrine clearly departs from the modern way of thinking at the EPO and, in a way, ignores all the developments and further steps taken by the EPO during the past decade. This is particularly due to Jacob LJ widening the computer programs exclusion to include programs 'in a practical and operable form', not just in the abstract.[67]

This approach differs from the view of the EPO, where the exclusion is interpreted in a narrow way and includes only computer programs as a set of instructions in an abstract way, and not where a reduction to a physical entity is present.[68] Additionally, under the 'any hardware' approach, the EPO will generally consider any application relating to computer implementations as an 'invention' for patentability purposes, while the UK-IPO will generally refuse to consider as an 'invention' an application whose contribution lies in a computer program. Further, when assessing non-obviousness, the EPO takes into consideration just those technical features of the claimed invention that contribute to the solution of the objective technical problem, rejecting computer-implementations of non-technical methods. The UK-IPO, in contrast, will consider any feature (whether technical or not) in assessing non-obviousness.[69] Finally, the response of the UK-IPO to *Aerotel/Macrossan* was flatly to reject any computer program claims, contrary to the EPO where such claims have been accepted since the *IBM* cases in 1999:[70] the vast majority of applications examined under the *Aerotel/Macrossan* test resulted in refusal, with most rejections being on the grounds that the claimed invention was a computer program as such.[71]

Although the original intention of Jacob LJ was to suggest a more harmonized and coherent approach for the interpretation of Articles 52(2) and (3) all over Europe, the formulation of such a discordant test represents the very opposite.

Recent developments

In July 2007, four companies (Astron Clinica, Software 2000, Surf Kitchen, and Cyan Holdings) appealed to the Patents Court of England and Wales against the rejection of six of their applications by the UK-IPO. All applications shared the same features of being claims for methods, devices, and software. Although the claims for

64 *Aerotel Ltd v Telco*, n 26.

65 ibid.

66 A Howes, 'Disaster Pending? EPO vs English Court of Appeal on Excluded Subject Matter', 08/07 World Intellectual Property Report.

67 *Aerotel Ltd v Telco*, n 26.

68 Salomon, n 45.

69 Bainbridge, n 66.

70 See 'technical effect' approach above.

71 This effect has lately been shown in some decisions based on the 'four-step' test, like *Cappellini/Bloomberg* [2007] EWHC 476 (Pat) and *Re Oneida Indian Nation* [2007] EWHC 0954, (Pat).

methods and devices were otherwise allowable, they were refused for involving computer program claims.

On the light of the interpretation of the 'four-step' test given by the UK-IPO and of its ensuing practice note,[72] one would have confidently predicted that the Court would dismiss the appeals. However, in *Astron Clinica*[73] Mr Justice Kitchin found that no elements of the *Aerotel/Macrossan* ruling support a blanket exclusion of all computer-related claims from patent law. He emphasized that *Aerotel/Macrossan* left open the question over the admissibility of such claims and that, accordingly, there was no element supporting the thesis that the 'four-step' test could not lead to the same outcome as at the EPO after the *IBM* cases. This being so, the four appeals were allowed and the applications remitted to the UK-IPO for further consideration in light of the judgment. Following *Astron Clinica*, the UK-IPO issued another new practice note,[74] re-amending the way the Office treats computer program claims.

Astron Clinica put the UK back on the same track as the EPO. However, whether this umpteenth turn of events equates British doctrine to the EPO's way of thinking or shed any light on the overall confused situation on software patentability in the UK remains questionable, as has been shown in two landmark High Court decisions in *Autonomy*[75] and *Symbian*.[76]

Autonomy raises strong doubts over the conformity of the UK practice with that of the EPO. Even if the computer program claims amendment was taken into consideration, the application of the *Merrill Lynch* 'technical contribution' approach led the Court to reject the application on the grounds that the contribution did not exist independently of the computer, did not require new hardware or combination of hardware, and did not result in a better computer. The only effect produced lay in the running of the computer program. Thus, the claimed invention was a computer program 'as such'. There is a strong reason to believe that the EPO 'any hardware' approach would have most likely brought to a contrary result.

Symbian questions the stability of British still further. The Patents Court's reversal of the UK-IPO rejection of *Symbian* application, together with the step taken by the Office to appeal against this decision[77] represent further evidence of deep uncertainty.

Germany

The requirements of the EPC, Articles 52(2) and (3) were transposed into the German Patent Act 1981[78] §1, last amended in 1998. Generally speaking, if the overall character of a program is technical and if it serves a technical purpose, it may be patentable under German practice. Non-technical programs, such as book-keeping or office organization programs, can never be patented under German practice,[79] which in practice is not fundamentally different from that of the EPO.

Interpretations of 'technical'

The meaning of 'technical' with respect to patent law was originally clarified by the Bundesgerichtshof (BGH, the German Supreme Federal Court of Justice) in *Rote Taube (Rote Tube)*[80] in 1969:

> patent protection is available for a method describing a methodical action using controllable physical forces to attain a visibly obvious effect or obvious success without interposition of human intellectual activity.

However, whether the *Rote Taube* definition eliminates anything from the group of technical inventions is questionable. Furthermore, the fact that the legislature never positively defined 'physical forces' makes that definition even less useful.[81]

As to computer programs, the BGH has sought to develop a practice that avoids relying mainly on the term 'technical' (unlike the EPO). It was thought that interaction between hardware (a technical device per se) and software would have made the 'technical' difficult to deny.[82] However, as transpires from some major

72 *Patent Act 1977: Patentable subject matter*, UK-Intellectual Property Office, http://www.ipo.gov.uk/patent/p-decisionmaking/p-law/p-law-notice/p-law-notice-subjectmatter-20080207.htm

73 *Astron Clinica Ltd and Others v Comptroller General* [2008] EWHC 85 (Pat).

74 n 76.

75 *Autonomy Corporation Limited v Comptroller General* [2008] EWHC 85 (Pat).

76 *Symbian Ltd v Comptroller General* [2008] EWHC 518 (Pat).

77 No decision has yet been take on the UK-IPO appeal. However, a statement confirming the Office intentions has been published and it is http://www.ipo.gov.uk/press/press-release/press-release-2008/press-release-20080318.htm

78 Bekanntmachung der Neufassung des Patentgesetzes (PatG), vom 16, Dezember 1980. Available in English at the 'Collection of Laws for Electronic Access' (CLEA) of the World Intellectual Property Organization (WIPO), http://clea.wipo.int

79 Schutz von Computerprogrammen, http://www.dpma.de/infos/schutzrechte/verfahren11.html

80 BGH GRUR 1969, 672 *Rote Taube*. Available in English in 1 *Int'l Rev. Indus. Prop. & Copyright L.* 136 (1970).

81 N Hoppen, *Software Innovations and Patents. A Simulation Approach* (Ibidem-Verlag, Stuttgart, 2005) 21–26.

82 This has been specified for instance in BGH Sprachanalyseeunrichtung. See also K-J Melullis, 'Some problems of patent law from a German viewpoin' on OJEPO Special Edition 2/2007, 13th European Patent Judges Symposium, Thessaloniki 12–16 September 2006, 184–199.

decisions, this was impossible to accomplish and, on various occasions, the term 'technical' was applied to software patent cases in a manner that closely resembles the modern EPO approach.[83]

Two different interpretative theories have been formulated in jurisprudence to clarify the concept of 'technical character' in software: (i) the core theory (*Kerntheorie*) and (ii) the cumulative reflection theory (*Gesamtbetrachtungstheorie*). The core theory (or 'material reflection theory') was used until the late 1990s: the content of the method claim was compared with the prior arts and only what was added by the method was taken into consideration when evaluating the technical character of the claim.[84] The reasoning behind this theory closely resembles the way of thinking of the EPO 'technical contribution' approach.

After 2000, however, the cumulative reflection theory started to take over, with a consequent radical diminution of the use of the core theory. Under the cumulative reflection theory, not only the solution but also the problem was considered for patentability purposes. It was later claimed that the most direct consequence of such an approach has been the clear increase in the number of software patents granted.[85]

The first decisions

The BGH expressed itself for the first time over the use of patent law for computer programs in 1976, in *Dispositionsprogramm* (*Disposition Program*).[86] The claims were to a process for calculating certain commercial results using electronic data processing equipment. The BGH doubted whether computer programs could be patentable and rejected the application. It was argued that 'an instruction for using the hardware of a computer in a new and unusual way was potentially patentable' and that, in order to have a technical nature, the instructions of the invention should have been directed 'to achieve a certain design, functionality or use of a computer'.[87]

That position was clarified by the same court a few years later, when it decided that an antilock brake system operated by a computer program was suitable for patent protection, in *Antiblockiersystem (Anti-blocking system)*.[88] Stating that not every program for a computer lacks technical character, the BGH affirmed that programs can use 'natural forces' in order to regulate technical processes and, if they do so, they can be suitable for patent protection.[89] Only when natural forces are used, and the computer and the program or algorithm are related to each other in a way that a technical result is achieved without human intellectual activity interposing, can the invention be patentable.[90] In other words, 'human intellectual activity' is not a physical force for patent purposes.

In these first decisions, the judicial tendency to afford software patent protection was quite restrictive. However, from the late 1980s, a change in the interpretation of the technical requirement resulted in the BGH more broadly accepting the patentability of computer programs in its jurisprudence.

Towards a more generous approach

A more liberal interpretation of what is 'technical' in software started to take place from the early 1990s. Three judgments stand out: *Tauchcomputer (Diving Computer)*[91] in 1992, and *Sprachanalyseeinrichtung (Speech Analysis Apparatus)*[92] and *Logikverifikation*[93] in 2000. These, together with the interpretative line that originated with them, marked the end of the 'core theory' and officially declared the beginning of what has been termed the 'cumulative reflection theory' which brought an increase in the number of software patents granted.

Tauchcomputer involved the application of a computer program for use in decompression time calculations for scuba diving.[94] This program did not control the machine, but rather the behaviour of the diver. Despite that, the patent was granted and the computer program considered technical as it processed, controlled, and displayed measured data. The importance of this judgment lies in the Court considering the technicality of all the features of the invention, not only of those that differed from the prior art. On this basis, considered 'as a whole', the automated data display required no human intellectual activity and was thus technical in nature.[95]

The possibility of obtaining software patents in Germany was expanded further by the interpretation given in *Sprachanalyseeintichtung* and *Logikverifikation*.[96]

83 Melullis, n 84.

84 Hoppen, loc cit.

85 ibid.

86 BGH GRUR 1977, 96–98 *Dispositionsprogramm*. Available in English in 8 *Int'l Rev. Indus. Prop. & Copyright L.* 558, 560 (1977).

87 ibid. See also Hoppen, loc cit. 38–52.

88 BGH GRUR 1980, 849 *Antiblockiersystem.*

89 U Loewenheim, 'Legal protection for computer programs in West Germany' *Berkeley Technology Law Journal*, http://www.law.berkeley.edu/journals/btlj/articles/vol4/Loewenheim/html/text.html

90 ibid.

91 BGH GRUR, 430 *Tauchcomputer.*

92 BGH 2000, 1007 *Sprachanalyseeinrichtung II*. Available in English in IIC Vol. 33 No. 3/2002, pp 343.

93 BGH, GRUR 2000, 498–499 *Logikverifikation.*

94 Above n 93.

95 ibid.

96 J Weyand and H Haase, 'Patenting computer program. New challenges', IIC, 2005, Issue 6, 647–663.

Sprachanalyseeintichtung was an application for a speech recognition system involving a computer program. The Bundespatentgericht rejected the application but the BGH stressed that the applicant had applied to patent a 'whole' system, not the computer program per se. In the court's opinion, any system involving industrial applications, including different switches and consuming energy, is technical; this is not altered by the type of software running in the machine used. In other words, when a patent application involves both software and hardware components, possibly non-technical software does not change the technical character of the hardware.[97] The system in question was not affected by the use of a processing text or human interaction, the invention remaining technical either way. Until this point, therefore, the BGH interpretation appears very close to that of the EPO. However, for the purpose of assessing the technical character, the Court considered it irrelevant 'whether the apparatus produces a (further) technical effect, whether technology was enriched by it or whether it made a contribution to the state of the art', finding it unnecessary to consider the 'technical effect' approach, at that time in use at the EPO, in its judgment.[98]

Logikverifikation concerned a computer program which simplified a part of the design process of integrated circuits. The Bundespatentgericht considered the invention unpatentable for lack of technicality but the BGH disagreed, in a decision that opened the possibility of patenting the whole field of design/production/layout software in Germany.

Again the Court stated that, in order to assess the technical nature of a computer program, one should look at all the circumstances. In this case, the fact that the software used an intellectual concept was irrelevant, because the overall goal was the realization of a technical idea.[99] Moreover (contradicting its earlier decisions and definitions of 'technical'), the BGH stated that even if no direct 'utilization of controllable natural forces without intermediate use of human intellectual power' was involved, the invention could be technical in nature if it was part of a larger technical production process. Thus, the BGH decided, following EPO practice, that to have technical nature, an invention should be based on a technical idea. In order to determine the 'technical character' of software, the focus should have been on the product designed and produced by the program, rather than on the program itself.[100] This reasoning reveals a clear departure from the original German practice, where the assessment of technicality was limited to the 'core' of the invention, in favour of a more liberal approach, where the invention is assessed as a 'whole'.

The court, conceding the inconsistency of this decision with its precedents, decided to amend the definition of 'technical' and consider it as a dynamic term, which could be interpreted differently in relation to the challenges presented by constant technological developments.[101]

Some thoughts

In Germany, the legal system has shown itself willing to confirm the patentability of certain program-related inventions when they have been of a technical nature, and when they have produced a technical solution to some technical problem.

The judgments cited above demonstrate that, gradually, the degree of 'technicality' required of an invention has been construed more generously and that the original impediment to software patentability, based on the 'technical nature' of an invention, has disappeared. Instead, the focus is now on the technical nature of the solution and the problem to be solved, ie on the inventive step. This is evidenced by important judgments such as *Sprachanalyseeinrichtung* and *Logikverifikation*. Gert Kolle, a leading German law scholar on questions of patentability of computer programs, was right in saying that 'any attempt to naively or willingly open up the patent system for computer programs would inevitably lead to unlimited patentability and to dangerous monopolizations of the sphere of pure reason'.[102] This trend is currently apparent in Germany.

The practice followed in Germany resembles that of the EPO. However, different interpretations have taken over in practice. This is particularly due to the difficulty in releasing simple, precise rules in the German jurisprudence that match those of the EPO, a point emphasized in *Sprachanalyseeintichtung* where the BGH felt obliged to ignore the EPO way of thinking in denying the application of the 'technical effect' approach. Another important reflection coming from the German experience is the clear difficulty in applying the requirement of technical nature to software, a difficulty that caused the BGH to embrace a flexible interpretation of such a requirement in *Logikverifikation*. Whether this position leads to a more transparent and coherent practice remains ques-

97 S Bechtold, 'Software patents in Germany. Current developments' (July 2000). Available at SSRN: http://ssrn.com/abstract=240205.

98 See *Aerotel Ltd v Telco* judgment, n 26.

99 Bechtold, n 99.

100 ibid.

101 Above n 95, para 8.

102 G Kolle 1977: Technik, Datenverarbeitung und Patentrecht—Bermerkungen zur Dispositionsprogramm—Entscheidung des Bundesgerichtshofs, GRUR 1977-02, 58–74.

tionable. Instead, what this situation emphasizes is a need for clarification, with a radical modification of the current rules.

France

France became the first country to adopt an explicit exclusion of software from patentability in its Patent Act 1968.[103] Although not without intense and contrasting discussions, this decision was taken mainly to prevent the already well-developed US software industry taking over the French market and securing an advantage over the still mainly start-up French companies.[104] However, in 1978, a revision of the Patent Act brought French patent law in line with the EPC and, in its current version,[105] Article L.611-10 reproduces Article 52 EPC. In addition, the doctrine specifies that the patentability of an invention should not be affected solely by it involving the use of an unpatentable element (eg software). The applications should, instead, be assessed in accordance with the patentable elements, ie the technical features of the invention.[106]

The early stage

Contrary to both the UK and Germany, French jurisprudence is quite limited and mainly refers to two, quite outdated, but still very important appeal cases.

The first relevant decision was *Mobil Oil*[107] arising from an application filed by the US company Mobil Oil for a system of choosing pigments through the use of a computer program. The court interpreted literally the exclusion of software from the scope of patent protection, affirming to be unpatentable not only a computer program as such but also any application of software (including the use of a process with an industrial application). *Mobil Oil*, however, was strongly criticized and the idea gradually prevailed that not all inventions involving computer programs were purely abstract: once a program had some industrial application, for example an industrial process controlled by the use of software, patentability was likely.[108]

The second important decision was by the Cour d'appel de Paris in 1981 in *Schlumberger*.[109] Here the court took a new, less restrictive approach, by openly stating that software could potentially be an object for patent protection: a patent should not be denied to a process merely because the invention involved the use of a computer program but, 'once the invention has an industrial character, it can indeed be protected under patent law'.[110] This way of thinking is very similar to the EPO practice.

The modern interpretation

The current French approach mainly conforms with EPO practice, EPO case law being officially incorporated into French jurisprudence and representing the basis for the formulation of principles and rules in this field.[111] However, the interpretation given by the French Institute for Intellectual Property (INPI) has resulted in the formulation of some more specific guidelines, which have often led to results diverging considerably from those at EPO level.

Generally speaking, a computer program per se, a program registered on a carrier, or an ordinary computer loaded with a computer program, are not patentable subject matter. However, as affirmed in *Schlumberger*, once an invention adds a 'technical contribution' to the previous art, it may not be excluded from patentability. Also, the processing of data representing physical parameters is likely to be patentable.[112]

French jurisprudence, like that of the EPO, considers 'technical character' a fundamental requirement. It is commonly understood that, for an invention to be patentable, a technical solution to a technical problem should be present. Non-technical elements are also sometimes allowed, but only if they are used to produce an overall technical result (or to solve a technical problem).[113]

Summarizing, France has sought to conform to the EPO way of thinking. However, its patent system is generally simpler and less strict than the EPO. This is particularly due to France (as compared with the EPO, British and German systems) lacking any substantial

103 Barzanò and Zanardo, *Brevetti e marchi* (2ª edizione, 1999) 109–120.

104 ibid. See also C Le Stanc, 'Logiciel: trente ans entre droit d'auteur et brevet. Bilan?', *Mélanges Linant de Bellefonds*, LexisNexis, Litec. (2007).

105 Arrêté du 19 septembre 1979 relatif aux modalités de dépôt des demandes de brevet d'invention et de certificat d'utilité et d' inscription au registre national des brevets.

106 P Mathely, *Le nouveau droit français des brevets d'invention* (1991) 128–140.

107 CA Paris 22 Mai 1973, PIBD 173. 107.III.197, Comm. 28 Mai 1975, PIBD 1975.155.III.

108 Réponse du groupe français de ÍAssociation Internationale pour la protection de la Propriété industrielle (AIPPI) à la consultation établie par les services de la direction générale du marché intérieur de la Commission européenne: brevetabilité des inventions mises en oeuvre, Rapport Q 133, http://www.industrie.gouv.fr/observat/innov/carrefour/aippi.htm

109 CA Paris 15 Juin 1981, PIBD 1981.185.III.175, Dossiers Brevets 1981.III.1.

110 ibid.

111 Above n 110.

112 ibid.

113 ibid.

pre-examination, which leads to a more generous approach in issuing patents. The only limit lies in section L. 612-12, 5° of the French intellectual property code, which allows the director of INPI to reject an application for an obvious lack of invention. This clause is rarely invoked and, broadly speaking, the INPI will never object to an invention clearly presented as a process to do something.[114] In this case, neither a patent issued by the EPO and designating France nor a patent filed solely in France will be rejected on the grounds of being a computer program as such.[115] As should be clear by now, this trend does not exactly conform to EPO practice.

Time for a change?

The practice to be followed regarding computer program patentability in Europe has been long discussed. Significant differences remain between EPO case law and national European jurisprudence. In this analysis, the leading jurisdictions of the UK, Germany, and France have been taken as examples to demonstrate that courts across Europe have different views concerning the patentability of computer programs. There is reason to believe that this feeling is shared by many other European countries.

A major problem deriving from the present fragmented system is the distortion of the European market, and the consequent impediment to economic growth. Companies operating in the sector will not be able to act equally throughout Europe and run the risk of seeing their products protected in some countries while being copied in others. This situation is detrimental not only for the companies themselves, but also for consumers, as it creates legal uncertainty.

In such a situation, harmonization may be desired. Harmonization would bring transparency and, in so doing, it would provide European companies with more incentives to apply for and use their patents for inventions involving software.

This was the goal behind the failed proposal of a European Union Directive on software patents,[116] rejected in 2005. However, a directive binding merely for EU countries would have solved the problem only partially, leaving the practice of both the EPO and non-EU EPC countries unchanged (the same objection may be raised with respect to COMPAT, the proposed European Community Patent).[117]

It appears, therefore, that the plan to create a unitary court competent to decide on patent infringement of the EPO Members, on the model proposed by the European Patent Litigation Agreement (EPLA),[118] would be a more efficient instrument for the elimination of the rifts between European patent laws in general. In the software field, a unitary court with judges specialized and specifically trained in the area would bring consistency to the interpretation of Article 52 EPC, leading overall to legal certainty.

Which direction harmonization should take is unknown. When applied to software, the technical requirement has repeatedly proven itself inappropriate and confusing. Since software patent protection is here to stay, the flawed basis upon which it stands requires repair. The time has come for European patent law to meet the need for protection of inventions in all fields of technology.

The way forward

In the context of harmonization, a special regime of interpretation should be applied to software patents. A reformulation of the traditional concepts of 'inventive step' is needed, together with the limitation of the scope of protection in software patents.

Jurisprudence and case law have demonstrated how assessing the scope of 'as such' under the concept of 'technical invention' creates confusion and uncertainty. This is why the EPO decided to change policy and admit computer implementations to 'invention' status. Although it appears that the seemingly absolutist 'as such' exclusion of computer programs from European patent law has currently lost any real meaning, this approach is undoubtedly an easier and more efficient way of interpreting European patent law without breaking the EPC principles. In addition, the EPO switch from 'invention' to 'inventiveness' represents an important step towards an innovative modern interpretation of patent law for computer programs.

One major problem remains. Under European patent law, inventive step is assessed with reference to the 'technical solution provided to a technical problem', taking

114 M Vivant and JM Bruguière, 'Protéger les inventions de demain', La Documentation Française 16 et seq. (Paris 2003).

115 See Ministere de la Justice, Universite Paris I ARPEJE, Colloque International Droit de linternet Approches europeennes et internationales, Rapport 'La Brevetabilité de Logicieles' (20 Novembre 2001) Alain Bensoussan, Avocats.

116 Proposal for a directive of the European Parliament and on the patentability of computer implemented inventions, Brussels, 20 February 2002, COM(2002)92 final (Software Patent Directive proposal).

117 For more information on the Community patent, see: http://ec.europa.eu/internal_market/indprop/patent/index_en.htm

118 'Draft Agreement on the Establishment of a European Patent Litigation System' (16 February 2004), http://www.european-patent-office.org/epo/epla/pdf/agreement_draft.pdf and 'Draft Statue of the European Patent Court' (16 February 2004),http://www.european-patent-office.org/epo/epla/pdf/statute_draft.pdf

merely technical, that is non-commercial, features into consideration for non-obviousness purposes.[119] The problem-and-solution approach[120] used by the EPO to address the inventiveness is an objective assessment made on a predominantly technical basis, which leaves the 'technical/non-technical' dilemma unsolved. The *Microsoft* judgment[121] is a clear example.

There is reason to believe that assessing the inventive step after considering all the features of the inventions would lead to a more coherent practice, avoiding the problems inherent in the current system. One advantage would be the elimination of the impossible task of separating technical, ie functional, features from non-technical ones, allowing examiners to concentrate their efforts on deciding whether the invention is actually non-obvious with respect to the current state of the art. Such interpretation would not go against the dictate of the EPC, which literally does not require any 'technical' element for the assessment of the inventive step. This unique interpretation of 'inventiveness' derives from German patent practice, under the 'advance test', which required 'advanced technical and economical utility', as well as novelty, for patentability.[122]

The different approach proposed here is not new since other jurisdictions, including the USA and Japan, have long been using it. However, in order to work efficiently for software, where innovation is relatively uncertain, the non-obviousness bar should be set quite high.[123] Generally speaking, in incremental industries, as is the case with software, a proper assessment of inventiveness prevents patent inflation, acting as a selective filter.[124] This suggests that, for software patents, the patent scope should be limited and patents should not extend to several product generations: this would stifle innovation in subsequent incremental developments, representing a huge hindrance for technological progress.[125] The Japanese patent system, where patent scope has traditionally been limited, provides a good example of how this would work. The narrow protection afforded under Japanese patent law has caused Japanese companies to develop a strategy of surrounding 'core' technology patents with many small minor improvement patents. In this way, the Japanese techno-industry has focused on small improvements over basic technologies imported from Western countries, such as the USA.[126] The risk behind this practice is an increase in the 'narrow' patents needed to commercialize a technology and a possible 'patent flooding', with consequent low quality patents, patent thickets, increased litigation, exclusionary conduct, and delay of cumulative innovation. To mitigate such problems in Japan, a system of patent pooling arrangements within corporations operating in the sector (under the culture of 'keiretsu') has been an efficient instrument.[127]

This practice clearly reflects the characteristics and structure of the software industry and suggests that the application of such a mechanism would be efficient in increasing innovation also in that field. Harmonization should, therefore, be addressed in this direction.

Inventive step is best

Patent law is an effective and relatively simple instrument to protect inventions by stimulating innovation. However, computer programs differ vastly from the products for which patent law was initially designed. Consequently, patents can be an efficient instrument of protection only once the system is adapted to cope with the software's special features. Extensive case law in the field proves that the 'technical requirement', main pillar of the traditional European patent system, as applied to computer programs, fails to meet its purpose. Continued reliance on such a practice is detrimental and inefficient.

A harmonized system, focusing mainly on the inventive step and with a relatively narrow patent scope would provide a reliable basis for software patent protection, eliminating the contradictions that currently exist with the application of the technical criterion. This would increase transparency and patent quality to the overall benefit of companies, consumers, and the European Community as a whole.

doi:10.1093/jiplp/jpn121
Advance Access Publication 10 July 2008

119 See 'problem-and-solution approach', EPO Guidelines (2007) n 8.

120 Composed by three steps: (1) the closest prior art is determined; (2) the technical problem is determined by comparing the results achieved in the invention with the closest prior art; (3) the obviousness of the solution is assessed in light of other art and the knowledge of a person with ordinary skill in the art (EPO Guidelines 2007).

121 See n 43 above.

122 K Saito and R Sweeney, 'Assessment of Inventive Step or Obviousness in United States, Europe, and Japan'. http://www.law.washington.edu/casrip/harmonization/PDF/obviousness.pdf

123 See Burk *et al.*, n 17.

124 ibid.

125 ibid.

126 See Saito *et al.*, n 123.

127 D Lin, 'Research versus development: patent pooling, innovation and standardization in the software industry', 1 *J: Marshall Rev. Intell. Prop. L.* 274 (2002).

ESSAY III

Rosa Maria Ballardini

The Software Patent Thicket: A Matter of Disclosure

(2009) 6:2 *SCRIPTed* 207

Peer reviewed

scripted

Volume 6, Issue 2, August 2009

The Software Patent Thicket: A Matter Of Disclosure

Rosa Maria Ballardini[*]

Abstract

The high complexity of software products, as well as the increased number of intellectual property rights in the field, has created a dense thicket of overlapping patent claims that companies must navigate in order to operate in the sector. The lack of relevant prior art and the abstract nature of the software patent claims are the major causes of overlapping patents in the field. However, efforts have thus far been concentrated merely in improving the prior art repositories. The abstract nature of the patent claims and the disclosure concerns deriving from that have, however, not yet received sufficient attention.

This article pursues this subject, first by investigating the reasons for, and consequences of, overlapping IP rights in software-related patents. This analysis suggests that overlapping problems and, thus, the software patent thicket, cannot be effectively reduced unless issues related to abstraction and disclosure are addressed. On the basis of this, a more detailed description of the programme – including flowcharts, pseudocodes and, when necessary, parts of the source code – might be an essential requirement to be added to the description of the invention in natural language.

DOI: 10.2966/scrip.060209.207

---

[*] Researcher in IP law at HANKEN School of Economics; member of the INNOCENT Graduate School in IP law, IPR University Centre, University of Helsinki; visiting scholar at UC Berkeley, Boalt Hall (2008/2009). The author thanks Professor Niklas Bruun and Assistant Professor Marcus Norrgård for their valuable comments. Sincere thanks also go to the telecommunications and computing team of the Finnish Patent Office, the participants of the DFG Graduate School n. 1148, Intellectual Property and the Public Domain Seminar at the University of Bayreuth, and the 8th Intellectual Property Scholars Conference at the Stanford Law School for commenting on earlier drafts of this paper. The usual disclaimer applies.

1. Software-Related Patents[1] and Patent Thickets

In incremental component industries like computer software – where new developments necessarily build upon existing technologies and innovation occurs through small improvements rather than real breakthroughs – inventions are usually highly complex and often protected by multiple intellectual property rights. Increasingly, products incorporate not just a single invention, but rather a combination of many different components, each of which may be the subject of one or more patents. Consequently, numerous licences might be required to produce even a single commercial product.[2] Furthermore, the vast proliferation of software-related patents in a relatively short period, together with the lack of relevant prior art in the field and the special nature of software as an abstract technology, has led to unoriginal, obvious, and vague patents being issued. This horizontal and dense overlapping of multiple patent claims is termed the "patent thicket."[3]

The term "patent thicket" originates from a litigation case in the 1970s regarding Xerox's dominance of a portion of the photocopier.[4] However, the economist Carl Shapiro has recently reintroduced it in the academic discourse, defining patent thickets as:

> *A dense web of overlapping intellectual property rights that a company must hack its way through in order to actually commercialize new technology.*[5]

Software patent thickets are mainly caused by two sets of problems: overlapping problems, related to the 'quality' of the patents, on the one hand; and problems related to the vast number of patents issued on the other. Both concerns might appear during the life-span of software-related patents, although at different stages. Specifically, the former set of problems represents the direct cause of the latter one.

This article pursues the subject, first by investigating the reasons for, and consequences of, the software patent thicket. Both the European and American perspectives are considered. In particular, the US's software patent landscape serves as a background for the discussion on the European situation.

The general concern revolves around thickets and software-related patents, and not around patent thickets within the software industry *per se*. It is important to keep this distinction in mind because the majority of software-related patents are obtained by

---

[1] In this paper the same meaning is given to both the term "computer programme" and "software" in the context of patent protection. Although the actual significance of "software" and "computer programme" differs, in fact, these terms can be used interchangeably under the assumption that the difference between the two is clear to all parties in the discussion.

[2] M Lemley and C Shapiro, "Patent Holdup and Royalty Stacking" (2007) 85 *Texas Law Review* 7, at 1991-2050.

[3] C Shapiro, "Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard–Setting" (2001) 1 *Innovation Policy and the Economy*, at 118-150.

[4] *SCM Corp. v. Xerox Corp*, 645 F. 2d 1195 (2d Cir. 1981) and *In re Xerox Corp*, 86 F.T.C. 364 (1975).

[5] See note 3.

firms operating outside the software industry.[6] As a consequence, the problem of software patent thickets, as it is here described, might not necessarily be reflected in the software industry. This does not mean, however, that software-related patents do not contribute to patent thickets in other sectors. Indeed, many of the industries that obtain most software-related patents – like semiconductor and computer industries – have been clearly identified by many researchers as having patent thickets.[7]

This article argues that the lack of relevant prior art and, in particular, the abstract nature of the software patent claims are the major causes of the overlapping problems in the field. However, although efforts have thus far been concentrated in improving the prior art repositories and various projects have been launched in order to reach this goal, the abstract nature of the software-related patent claims and the disclosure concerns deriving from that have not yet received sufficient attention. The thesis pursued here argues that overlapping problems, and, thus, the software patent thicket, cannot be effectively reduced unless issues related to abstraction and disclosure are addressed.

On the basis of this, a more detailed description of the programme – including flowcharts, pseudocodes and, when necessary, parts of the source code – might be an essential requirement to be added to the description of the invention in natural language. Enhancing disclosure would reduce the abstract nature of the software patents claims, consequently providing support both for better defining the boundaries between patentable inventions and prior art; and for narrowing the scope of the patents granted. Increased notice would reduce the number of the applications filed and thus also indirectly smooth the thicket.

Such a policy might be essential, not only for patent officers to better evaluating the applications, but also to support judges in dealing with questions of infringement. Competitors and new inventors – who necessarily need full access to the patented inventions in order to improve upon it – would also clearly benefit from such a practice.

2. Overlapping Problems In Software Patents

Overlapping problems represent a major cause of the software patent thicket. For the purposes of the present article, overlapping problems refer to patents that either should not have been issued, because the invention was not new or not unobvious; or patents that, even if patented on inventions that fulfilled the patentability requirements, should have been granted a much narrower scope of protection. As is evident, overlapping problems might arise either during the examination phase or when a patent is challenged in courts where interpretation of the patent's scope is needed.

Probably the most memorable example of an unoriginal software-patent is the Amazon.com patent, granted by the USPTO in 1999.[8] The patent, nowadays known as

---

[6] J Bessen and R Hunt, "An Empirical Look at Software Patents", 16 *Journal of Economics and Management Strategy* 1, at 157-189.

[7] See J Bessen and M Meurer, *Patent Failure* (Princeton University Press, 2008), at 187-214.

[8] US patent 5,960,411. See also *Amazon.com v Barnesandnoble.com* [C99-1695P], Seattle District Court (12/01/1999).

Amazon's "1-click" patent, covered an online system that allowed customers to enter their credit card numbers and address information just once so that, on follow up visits to the website, all it would have taken to make a purchase would have been a single mouse-click. Although it is still not clear whether Amazon was indeed the first to implement such a purchasing process, it is evident that the idea behind the invention was extremely simple and obvious to the eyes of a person skilled in the art. This lack of inventiveness was, for instance, the basis of the revocation of Amazon's "Gift Order Patent" application by the EPO Opposition Division.[9]

Another remarkable example of "trivial" software patent is the Wang's patent, obtained in the US in 1988.[10] The patent claimed, among other things, the general use of "frames" to display different types of information retrieved from servers. Clearly this was a very broad patent that claimed a vast range of technologies that might use a graphical user interface – far beyond Wang's actual product. Immediately the issued-patent raised concerns among Internet companies. It was only following a lawsuit by Wang against Netscape that a court decision narrowed the scope of the patent by interpreting the term "frame" restrictively.[11]

Overlapping software patents are not, as is often thought, merely an American concern, but rather a global issue. Bergstra and Klint,[12] for instance, eloquently highlight how in Europe overlapping problems are also strongly felt in the computer programming sector.[13]

Within the examples, they cited a patent "Apparatus for handling tag pointers," granted to IBM,[14] and describing the addition of a tag-bit to pointers in order to discriminate them from ordinary data, which represented a clearly old technique used in various systems long before the filing of that patent application. Examples of this include a patent granted to Sun[15] for a method and apparatus for simultaneously displaying graphics and video data on a computer display, which was a common technique that had been in use for many years before the patent filing; and the patent "Data pre-fetch for script-based multimedia systems,"[16] granted to Intel in 2000, aiming at speeding up the execution of multimedia scripts running in a limited memory client by pre-fetching data references that occur in the script, which was an obvious, non-inventive technique.

As these examples show, Bergstra and Klint's study considered primarily applications from large companies such as Microsoft, IBM and Sun. Allegedly, these companies have large patent practices and sufficient resources to determine whether an invention is actually inventive and to identify prior art before filing a patent application. Thus it

---

[9] EPO opposition hearing regarding "Amazon gift ordering patent" EP 0927945 B1 (07 Dec 2007).

[10] US Patent 4,751,669: "Videotex Frame Processing."

[11] *Wang Lab, Inc. v America Online, Inc*, 197 F.3d 1377 (1999).

[12] J Bergstra, and P Klint, "About 'Trivial' Software Patents: The IsNot Case" (2007) 64 *Science of Computer Programming* 3, at 264-285.

[13] *Ibid*.

[14] EP 10186.

[15] EP 752695.

[16] EP 767940.

can be argued that the examples analysed by Bergstra and Klint represent potentially typical, rather than unusual, applications.[17]

It is evident that overlapping software patents might lead to several negative consequences – the "dense web" of intellectual property rights being the most direct one. Low patentability standards, in fact, inevitably lead to a vast number of patents being granted. The radical increase in the number of patents issued in the software field is clearly evidenced by recent statistics both in the US and in Europe.[18]

Other reasons of concern include uncertainty, both on the value of the property right and on the validity of the scope of the right;[19] high transaction costs, due to complicated licensing negotiations, the risks of "royalty stacking", "hold ups,"[20] and "blocking" patents;[21] and risks of infringement, litigation, patent trolls and threats of injunctions.[22]

To address the issues mentioned above, the thesis argues that overlapping software patents find justification mainly in the lack of good prior art repositories and, in particular, in the intrinsic abstract nature of the software patent claims. Both perspectives need to be taken into consideration in order to solve the problem.

2.1 Reasons For Overlapping Software Patents

2.1.1 Lack Of Relevant Prior Art

One of the main reasons for overlapping software patents is the lack of relevant prior art in the field. Judging the novelty and originality of the inventions without an adequate prior art search is clearly a highly challenging task. Two different problems should be distinguished: lack of knowledge due to the difficulty of gathering prior art material on the one hand; and lack of knowledge due to prior art not being publicly available on the other.

The first aspect refers to cases where the prior art, even when in printed form, is impossible or very difficult to be found. First, much of the prior art in software technology lies outside the areas where patent examiners traditionally look – i.e. printed resources, such as domestic and foreign patents, and published literature.[23]

---

[17] See note 12.

[18] For Europe see EPO Annual Report 2007 - Statistics (while the total number of patents granted has decreased with respect to the previous years, the fields of electronics and computing have continued to grow). See also "World Patent Report. A Statistical Review", WIPO (Ed. 2008).

[19] M Lemley and C Shapiro, "Probabilistic Patents" (2005) 19 *Journal of Economics Perspectives* 2, at 75-98.

[20] N Thumm, "Blocking Patents and Their Effects on Scientific Research: Evidence from the Biotechnology Industry" (2005) *IPR Helpdesk Bulletin*, No 23.

[21] R Merges, "Intellectual Property Rights and Bargaining Breakdown: the Case of Blocking Patents" (1994) 62 *Tennessee Law Review* 1, at 75-106; M Lemley, "The Economics of Improvements in Intellectual Property Law" (1997) 75 *Texas Law Review* 5, at 989-1084.

[22] Recent empirical studies in the US clearly show that litigation risks are particularly high in the software (and business methods) field. See note 7.

[23] J Park, "Evolution of Industry Knowledge in the Public Domain: Prior Art Searching for Software Patents" (2005) 2 *SCRIPT-ed* 1, at 47-70.

Some inventions, in fact, solely appear in textbooks or user manuals that are usually not available to the patent examiners.[24] Additionally, because multiple representations of the same inventions are available in software technology, it is highly challenging to draw the boundary between allegedly different inventions, as discerning which specific piece of prior art is relevant for the claimed invention might be very difficult.[25] The drawing of such a boundary, however, is essential in order to assess the inventions in terms of prior art. Finally, the use by many patent attorneys of very broad terms so as to avoid a patentability objection in software patent applications, has rendered it hard to find the pertinent pieces of prior art.[26] The abstract nature of software patent claims clearly exacerbates these challenges.

The second aspect of the problem relates to the fact that many well-known software techniques have never been disclosed, in printed form or at all. Although they might be used in the source code of many software systems, they do not appear in any scientific publication.[27] Moreover, some software inventions are merely incorporated into products. For instance, methods for doing business (which are most of the time implemented through the use of a computer programme) are not available in journals, libraries or by searching databases, but rather in the web-material and the business plans of companies.[28]

Particularly relevant is also the fact that software was not considered patentable subject-matter up until the end of the 1990s and, as a result, many software-related inventions were (and still are) protected by trade secrecy, and hence remain undisclosed. It was only after the *Diamond v Diehr* decision[29] in the US and the Vicom decision[30] at the EPO that computer programmes started to be partially accepted for patentability purposes.[31]

Some restrictions still currently apply in Europe, with the European Patent Convention (EPC) excluding computer programmes "as such" from the patentability field,[32] and the European Patent Office (EPO) accepting only "technical" inventions for patent law purposes.[33] It is worth mentioning that limitations to software

---

[24] *Ibid*.

[25] See note 7.

[26] M Lemley, "Rational Ignorance at the Patent Office" (2000-2001) 95 *Northwestern University Law Review* 4, at 1495-1532.

[27] See note 23.

[28] J Cohen, "Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implications of 'Lock-Out' Technologies" (1994-1995) 68 *Southern California Law Review* 5, at 1091-1202.

[29] *Diamond v Diehr*, 450 U.S. 175 (1980).

[30] T0208/84 *Computer Related Invention/Vicom* [1987] OJEPO 14.

[31] J Newman, "The Patentability of Computer-related Inventions in Europe" (1997) 19 *European Intellectual Property Review* 12; C Laub, "Software Patenting: Legal Standards in Europe and the US in View of Strategic Limitations of the IP Systems" (2006) 9 *Journal of World Intellectual Property* 3, at 344-372.

[32] Convention on the Grant of European Patents of 05 October 1975 (European Patent Convention), Article 52, paragraphs (2) and (3).

[33] See EPO, Guidelines for Examination, Part C, Ch 4 (2007).

patentability might also be imposed in the US following a recent Court of Appeal, Federal Circuit's decision in the *In Re Bilski* case.[34] Even though the decision mainly addresses business methods patents, restrictions to software patents in general might ensue depending on the interpretation of the "machine or transformation" test formulated in *Bilski*. The Supreme Court has now granted Bilski a writ of certiorari, thus more clarity on the proper application of the test is expected later in 2009.

Altogether, the features mentioned above have rendered it difficult to keep most established publications up-to-date with many new developments in the computer programming arena. Accordingly, patent officers have found it challenging to keep non-patent databases updated[35] and to find relevant prior art references when searching. Patent officers' ignorance of relevant patent literature has lead to many patents being issued on subject-matters already in the public domain.

2.1.2 Abstraction

Patent law traditionally does not protect abstract ideas or principles, but rather practical and tangible implementations that might derive from such ideas or principles. Software patents applications, however, often include abstract claims. This causes at least three major problems. Firstly, abstraction leads to technologies being claimed for that are often not yet known to the inventor at the time of the application. As a consequence, patents might be issued very broadly and inventors be rewarded for things they did not invent. Secondly, abstraction makes it hard for the examiners to judge over the sufficiency of disclosure of the inventions and thus to decide whether enough information has been provided to properly support the applications. Finally, such a configuration makes it difficult for patent officers to draw the boundary between allegedly new and inventive inventions, and prior art. The abstract nature of software patent claims, in fact, leads to the issuing of patents that do not possess clear boundaries and, thus, gives rise to high risks of infringement and opportunistic litigation.[36]

It is worth noticing that infringement risks do not only affect software companies. On the contrary, every company that uses a website, accounting database and some in-house software specialist to design and customise the system, can find themselves sitting "on the wrong side" of a patent infringement suit.[37] A clear example of such a configuration is *the Global Patent Holdings LLC v Green Bay Packers* case. Global Patent Holdings had a patent on the use of certain images on a web site. Global Patent Holdings sued companies such as CDW Corp, Motorola, the Green Bay Packers, Caterpillar, etc., seeking settlements between 7 and 15 million USD. None of these was a software company.

---

[34] *In Re Bilski*, F.3d, 2008 WL 4757110, 88 U.S.P.Q.2d (BNA) 1385 (Fed. Cir. Oct. 30, 2008).

[35] A detailed list of existing non-patent databases is available at the EPO, USTPO and JPO, see Park at note 23 above.

[36] See note 7 above, ch 9-10.

[37] *Global Patent Holdings LLC v Green Bay Packers, No 00 C 4623, and Global Patent Holdings LLC v CDW Corp*., No 07 C 4476.

Litigation risks are particularly high in the United States, where recent studies have shown that software patents are responsible for a major share of the patent lawsuits, playing a central role in the failure of the patent system as a whole.[38]

In Europe, instead, the litigation activity still remains relatively low.[39] This is particularly due to the fact that no business methods *per se* and fewer weak software-related patents are granted in Europe. However, patent thickets clearly exist also in Europe and opportunistic behaviour might arise because of the currently unregulated use of patents. In particular, the creation of a common European litigation system under the proposal of the European Patent Litigation Agreement (EPLA),[40] or the Community Patent (COMPAT),[41] as it stands, raises serious concerns on the exacerbation of patent thickets. This issue is explained in more details later.

2.1.2 Software-Related Inventions v "Traditional" Inventions

The challenges raised by abstraction when assessing patentability of software-related inventions appear quite evident when comparing "traditional inventions" (i.e. physical and/or tangible inventions, closer to those that the patent system was originally designed for) with software-related inventions.

As an example of "traditional invention" let us consider a European patent granted by the EPO for a machine for cutting and splitting wood.[42] The machine takes timber wood as input and produces wood cut in pieces as output. The summary of the invention in the application stated:

> *A chopping machine for cutting and splitting a timber, said chopping machine comprising a crosscutting device for cutting the timber, a feeder for feeding the timber in its longitudinal direction to the crosscutting device, a splitting apparatus operated by a splitting cylinder for splitting a cut block of timber, said feeder comprising two elongated supporting surfaces forming a substantially horizontal trough open in the upward direction, into which the timber to be treated can be placed…*

---

[38] See note 7, at 120-146.

[39] See EU Commission, Internal Market, Patent Litigation Insurances Studies, at: http://ec.europa.eu/internal_market/indprop/patent/index_en.htm (accessed 25 Feb 2009). The study shows that in 2006 only four EU countries had had more than 100 patent cases (including all fields of technologies) per year, fourteen countries had had less that ten, and about half of the EU's member states had not had any case for many years.

[40] "Draft Agreement on the Establishment of a European Patent Litigation System," at: http://documents.epo.org/projects/babylon/eponet.nsf/0/B3884BE403F0CD8FC125723D004ADD0A/$File/agreement_draft_en.pdf (accessed 25 Feb 2009) and "Draft Statute of the European Patent Court", at: http://documents.epo.org/projects/babylon/eponet.nsf/0/885CCB85F5CC33ABC125723D004B15F9/$File/statute_draft_en.pdf (accessed 25 Feb 2009).

[41] For more information on the Community Patent see: http://ec.europa.eu/internal_market/indprop/patent/index_en.htm (accessed 25 Feb 2009).

[42] EP 1118438.

Let us now consider the computer-implemented invention claimed in *Vicom*.[43] In *Vicom*, the claims were both for a method of digitally processing images and for an apparatus for carrying out the method. The image was inputted as pixels and the processed image outputted as pixels. In other words, it was a digital signal processing system for image processing. The description of the invention mainly lied in Claim 1 and 8 (after the amendments filed on appeal), which read as follows:

> *1. A method of digitally processing images in the form of a two-dimensional data array having elements arranged in rows and columns in which an operator matrix of a size substantially smaller than the size of the data array is convolved with the data array...*
>
> *8. Apparatus for carrying out the method in Claim 1 including data input means for receiving said data array, and said data array to generate an operator matrix for scanning said data array to generate the required convolution of the operator matrix and the data array, characterised in that there are provided feedback means for transferring the output of the mask means to the data input means, and control means for causing the scanning and transferring of the output of the mask means to the data input means to be repeated a predetermined number of times.*

In the first example the machine had clearly physical implementations: the components were concrete (e.g. crosscutting device, feeder, splitting apparatus, etc.); and the functionality was easily comprehensible (also to persons not necessarily skilled in the art). Conversely, in Vicom, not only the did the components not have any physical implementation, but also the final result was intangible. Additionally, the language used in the application is clearly much more obscure. Overall, this configuration suggests that assessing patentability of software-related inventions might generally be much more challenging than for "traditional inventions."

Plotkin[44] also provides clear evidence of how assessing patentability of software-related inventions implies more interpretation than in other fields of technology. Plotkin addresses the issue by comparing software-related inventions with other electromechanical inventions in an attempt to demonstrate the extent to which the software's special qualities should be reflected in intellectual property laws.

He focuses in particular on the fact that patent law traditionally requires electromechanical device's components "to be conceived of, described, and claimed in terms of their physical, not merely logical, structure." In other words, for electromechanical devices, as well as a functional design, the provision of a physical structural design is hard and essential for obtaining patent protection. Conversely, in software, the logical structures described by the source code represent the end point of the invention. The step to their physical realisation is, instead, fully automated.[45]

---

[43] EP79300903.

[44] R Plotkin, "Computer Programming and the Automation of Invention: A Case for Software Patent Reform" (2003) 7 *UCLA Journal of Law and Technology* 2. Available at SSRN: http://ssrn.com/abstract=503822 (accessed 25 Feb 2009).

[45] *Ibid*.

The situation thus far described is worsened by the fact that software patent applications generally contain a higher number of total and independent claims than other types of patents.[46] Software patent applicants, in fact, are more likely than others to claim inventions in duplicative ways within a given patent. Software-related patents, for instance, often include sets of claims that characterise the invention as a method (or process), a machine or apparatus (or device), and a system. This is mainly because software inventions can be conceptualised in many different ways. The Blackboard patent on an "Internet-based education supporting system and method" granted by the USPTO[47] and recently litigated in the United States,[48] for instance, represents an outstanding example of such a configuration.[49]

The Blackboard's patent protected an "Internet-based education supporting system and method." In other words, it covered the entire e-learning process, including methods, virtual learning environments (VLEs), discussion forums, delivering classes via webcast or podcast, and just about everything else.

Considering the extreme broadness of this patent, it is not difficult to imagine that Desire2Learn, a direct competitor as well operating in the manufacturing of educational software business, was accused of infringement soon after the patent was granted. Subsequently, a jury in Lufkin, Texas found for the plaintiff on issues of literal infringement, infringement by equivalents, induced infringement and contributory infringement of the patent, and awarded Blackboard a total amount of 3,265,633.00 USD. A permanent injunction against Desire2Learn was then issued.

2.1.3 Conclusions

The analysis above shows that it is generally much more difficult to evaluate the concrete applicability of inventions containing abstract claims. Thus, more interpretation is required in order to assess the question of patentability. An extensive disclosure, therefore, would seem to be necessary in software patent applications in order to ease the process.

Surprisingly, however, disclosure in software-related patents is usually quite poor and these patents generally reveal very little of the inventions they protect. As a result, all the mentioned problems have been particularly accentuated – patents have been given too broad a scope of protection and patentees have been rewarded for things they did not invent. This has, overall, contributed to the intensification of the problems of overlapping patents in the field.

These aspects play a particularly important role for the purposes of the present article. As will be explained in the following sections, in fact, there is reason to believe that abstraction and, to a certain extent, prior art concerns could be reduced by promoting increased disclosure in the applications.

---

[46] See note 7.

[47] US patent 6,988,138.

[48] *Blackboard Inc. v Desire2Learn Inc*. [9:2006cv00155], Texas Eastern District Court, Lufkin Office (26 July 2006).

[49] *Blackboard Inc. v Desire2learn Inc*. [9:06CV155], Texas Eastern District Court, Lufkin Office, "Judgment and Permanent Injunction" (11 March 2008).

2.2 Disclosure Of Information In Software-Related Patents

2.2.1 The Requirement of Disclosure

The term "patent" derives from the Latin word patere which means "open letter."[50] In other words, it means to make the invention available for patent inspection. The requirement of disclosure in patent law, therefore, constitutes the tool to ensure the "openness" of the inventions.

Disclosure is a clear trade-off under which the imposition of a monopoly to society is justified by the assumption that the benefit deriving from it (i.e. further technological progress and innovation) is greater than the negative effects of those temporary restrictions.[51] Therefore, a patent represents both an incentive and a reward that the society gives for the disclosure of new information that might ultimately lead to more technological wealth.

In today's society the disclosure requirement is not only seen as a means to enable the public to reproduce the invention, but also as information that can be used to both improve on a specific invention and permit the conducting of further research in other fields of technology.[52] Additionally, by spreading technical knowledge, disclosure contributes to a more efficient allocation of resources.[53] Finally, a proper disclosure should enable the examiner to delimit clear boundaries within novel and original inventions, and prior art, which, in this way, should prevent the overlapping of patent rights.

In Europe, Article 83 EPC states that "the European patent application shall disclose the invention in a manner sufficiently clear and complete for it to be carried out by a person skilled in the art". This means that an application must contain sufficient information so as to allow a skilled man, using his common general knowledge, "to perceive the technical teaching inherent in the claimed invention and to put it into effect accordingly."[54]

Written description, enablement and best mode are all fundamental pre-requisites for obtaining a patent under US patent law. Section 112 of the US Patent Act, in fact, requires patent applicants to describe the invention in a manner sufficient to enable one of ordinary skill in the art to make it and use it, as well as setting forth the "best mode" to implement the invention.[55] Hence, generally speaking, sufficient disclosure plays a fundamental role for the issuing of "valid" patents. Some conditions specific to the software field, however, impose restrictions on the disclosure of software-related patents.

---

[50] G Friedrichsen, R Burchfield, and C Onions, *The Oxford Dictionary of English Etymology* (Oxford: OUP 1996).

[51] D Chisum et al, *Principles of Patent Law. Cases and Materials* (New York NY: Foundation Press 2004), at 49-51.

[52] *Ibid*.

[53] W Landes, and R Posner, *The Economic Structure of Intellectual Property Law* (Harvard University Press 2003), at 328.

[54] EPO Enlarged Board of Appeal Decision G 2/95 (21 Dec 1994) and G 2/98 (21 May 2001). See also EPO Board of Appeal Decision T1191/04 (22 Nov 2007).

[55] 35 U.S.C. §112.

2.2.2 The Software Patents' Restrictions

The special nature of software technology, as well as court decisions and legal dispositions, altogether have contributed to drastically curtailing the disclosure requirements in software-related patents. Disclosure concerns are particularly felt in the United States, where would-be patentees of software inventions are generally not required to disclose much. Case studies[56] have shown how a series of Federal Court decisions have drastically curtailed the enablement and best mode requirements by holding that patentees do not need to disclose source or object code, flowcharts or detailed descriptions of the patented programme, while finding high-level functional descriptions sufficient to satisfy disclosure requirements.[57]

In Europe, the EPC and its related dispositions impose a somehow higher disclosure threshold. Generally speaking, for computer-implemented inventions (CII) (that is, inventions for which the implementation involves the use of a computer, computer network, or other programmable apparatus – the invention having one or more features that are realised wholly or partly by means of computer programmes),[58] applicants must provide a description in natural language, whereas disclosing the invention through programming languages is considered insufficient.[59] In addition, applicants might provide flow diagrams and other examples to enable better understanding of the invention.[60] In Europe, however, neither the disclosure of the designing documents nor the source code of the programme behind the patented CII is necessary to fulfil disclosure requirements.[61]

Both in the US and in Europe, therefore, any obligation for patentees to disclose information of the programme behind the invention has been merely nullified. In such an abstract technology as computer software, however, this obscure configuration might be especially detrimental for promoting further progress.

Notably, competitors and new inventors need access to the programme (lying behind the invention) in order to reproduce the patented invention, build upon it and thus create further developments. As computer programming is a highly technical and difficult art, in fact, pretending that one with ordinary skills would be able to reconstruct a software invention, giving no more than the function the programme is

---

[56] D Burk and M Lemley, "Designing Optimal Software Patents", in R Hahn (eds) *Intellectual Property Rights in Frontier Industries: Software and Biotechnology* (AEI Press 2005).

[57] See, for example, *Fonar Corp v General Electric Co*, 107 F.3d 1543, 1549 (Fed. Cir. 1997) and *Northern Telecom, Inc v Datapoint Corp*, 908 F2d 931 (Fed. Cir.), cert. denied, 111 S. Ct. 296 (1990).

[58] See "Patents for Software? European Law and Practice" (2008) European Patent Office. Available at: http://documents.epo.org/projects/babylon/eponet.nsf/0/a0be115260b5ff71c125746d004c51a5/$FILE/patents_for_software_en.pdf (accessed 25 Feb 2009).

[59] *Ibid*. See also EPO, Guidelines for Examination, Part C-II, 4.15 (2007).

[60] *Ibid*.

[61] For instance, the EPO has recently issued a new brochure on the patentability of CII explicitly specifying that the disclosure of the inventive concept "does not require disclosure of a source code." See European Patent Office, "Patents for Software? European Law and Practice" (2009). Available at http://documents.epo.org/projects/babylon/eponet.nsf/0/a0be115260b5ff71c125746d004c51a5/$FILE/patents_for_software_en.pdf (accessed 11 June 2009).

to perform, seems highly unrealistic. However, since the source code is usually kept secret and programmers disclose very little information of the programme behind the inventions in the patents, reverse engineering might be necessary to reveal what the patentee has invented.[62]

Although this configuration is obviously not a unique problem of software-related patents, in such a field the consequences might be particularly detrimental due to the possible problems associated with decompilation.

2.2.3 Trouble with Reverse Engineering Software

Reverse engineering is an important practice to access technical information in many fields of technology. Inventors often need to reverse-engineer patented products in order to study and understand the technology so as to improve upon it. In the software field, reverse engineering through "decompilation" involves working backwards from the binary object code to produce a simulacrum of the original source code.[63] Although, theoretically, reverse engineering might be a successful tool to reveal the knowledge hidden behind patented software-related inventions, a series of both legal and technical reasons might block such an accomplishment.

From a legal perspective, software decompilation might be forbidden under certain conditions. Lemley and Cohen[64] have argued that because decompilation involves the generation of a copy of the patented programme, it might fall within the broad category of conduct prohibited by patent law – that is "making, using, selling and importing" the product. Specifically, decompilation might constitute "using" and "making" the patented programme by generating a temporary yet functional copy of it in the RAM memory.[65] On this issue, however, the European and American traditions differ.

In most European jurisdictions reverse engineering software through decompilation falls under the research privilege as "experimenting on."[66] Under the American practice, however, where reverse engineering is accepted only for non-commercial purposes, such an exemption might not apply. Nor would the doctrine of exhaustion suffice in solving the problem. The doctrine of exhaustion makes a specific distinction between "using and reselling" a particular copy of a patented product (permissible) and "making" a new copy of a patented product (not permissible).[67] The fact that decompilation constitutes "making" the patented programme might be found to be infringing patent law. The problem specifically arises with respect to product patents,

---

[62] See note 56. See also P Samuelson et al, "A Manifesto Concerning the Legal Protection of Computer Programs" (1994) 94 *Columbia Law Review* 8, at 2308-2432.

[63] A Johnson-Laird, "Software Reverse Engineering in the Real World" (1994) 19 U. Dayton L. Rev. 3, at 843-902.

[64] See M Lemley, and J Cohen, "Patent Scope and Innovation in the Software Industry", 89 *California Law Review* 1. Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=282790 (accessed 25 Feb 2009).

[65] *Ibid*.

[66] For more information see "Research Use of Patented Knowledge: a Review", STI Working Paper 2006/2, OECD Directorate for Science, Technology and Industry (STI).

[67] See note 56.

while for process patents the ordinary "use" of the process is justified either under principles of exhaustion or implied licence.[68]

Even when reverse engineering is considered lawful, such a process might incur technical problems that might render it highly inefficient.

Software is usually distributed in object code form. Decompiling object code is not only highly expensive and time-consuming, but it is also a very difficult and extreme way of examining the structure of software applications.[69] This is particularly felt in modern software technology where specific tools are commonly used to make reverse engineering very challenging. Furthermore, the fact that compilation from source code to binary code always loses information and comments written by the programmers, often renders decompilation quite useless.

Overall, it can be concluded that reverse engineering should not be a necessary tool to replicate software-related inventions. Firstly, patent disclosure per se does not imply access to any artefact in order to reveal the secrets behind patented inventions. Secondly, reverse engineering is not a very useful tool in software: on the one hand it merely enables the understanding of the implementation, not the design, of the computer programme, which is better conveyed through higher-level means; on the other, the more developments rely on sophisticated tools to produce code automatically (different tools producing different codes for the same specification), the less useful reverse engineering becomes.[70]

2.2.4 Conclusions

The intangibility of software-related inventions, as well as the difficulty to discern information from those types of patents, call for a more extensive disclosure policy. On the one hand, enhanced disclosure is important in order to reduce one of the major causes of overlapping software patents, i.e. the abstractness of the claims; on the other, it might also be essential for software-related patents to meet patent law purposes. Patents are a quid pro quo under which the imposition of a monopoly is justified by the assumption that the society would be better off under this restriction due to the benefit deriving from the disclosure of new technological information. On these grounds, the failing to meet a sufficient 'disclosing threshold' might well be considered unlawful.

3. Trends In Legislation: An Overview

As technology becomes increasingly complex the patent thicket is accentuated. This trend is particularly reflected in highly technical fields, such as computer programmes.

---

[68] *Ibid*.

[69] See A Johnson-Laird, "Reverse Engineering of Software: Separating Legal Mythology from Actual Technology" (1992) 5 *Software Law Journal* 2, at 331-354. See also P Samuelson, and S Scotchmer, "The Law and Economics of Reverse Engineering" (2002) 111 *Yale Law Journal* 7, at 1575-1664.

[70] See M Campbell-Kelly, and P Valduriez, "A Technical Critique of Fifty Software Patents" (January 2005). Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=650921 (accessed 25 Feb 2009).

Recently, legislators on both sides of the Atlantic have started to realise the potentially devastating effects patent thickets can bring to innovation, and awareness has been raised at the patent offices and the national courts. This trend highlights that the software patent thicket is clearly a global concern and, as such, global solutions rather than national-based proposals should be promoted.

It should be stressed that, even though problems related to patent thickets in abstract technologies have now been openly recognised, attention has thus far been posed merely on issues of patentable subject-matters, novelty, non-obviousness and use of injunction. The disclosure requirement, instead, has not yet been properly considered as a remedy, neither by the courts nor by the legislators.

3.1 US Developments

In the United States, the problem of patent thickets has recently caught the attention of much of the scientific and engineering community in a number of technological arenas.[71] The 2003 Federal Trade Commission (FTC) report,[72] for example, emphasises how in certain industries the large number of patents issued makes it virtually impossible to search all the potentially relevant patents; review the claims contained in each of these patents; and evaluate the infringement risks or the need for licences. The Commission cites hold–up problems for the software sector, in particular, by pointing out that "the owner of any one of the multitude of patented technologies constituting a software program can hold up production of innovative new software."[73]

This wave of criticism led both the legislator and the courts to initiate different plans of action to tackle the problem. This is reflected, for instance, in the Strategic Plan[74] launched by the USPTO to improve the general patent landscape, and on the heated debates on issues of patent reform. Additionally, there has been a renewed interest by the courts in general, and by the Supreme Court in particular, towards patent cases.[75] The need to both curtail patentable subject-matter and curb patent rights emerges from some recent decisions in the software and business method patents sector.

eBay v MercExchange,[76] for instance, was a business method case which limited the use of injunctions in infringement cases and, thus, represented a clear effort of the Supreme Court to tackle one of the most direct consequences of patent thickets, i.e. patent trolls.[77] The *KSR Intern. v Teleflex* decision[78] is another clear attempt by the

---

[71] See G Clarkson, "Cyberinfrastructure and Patent Thickets: Challenges and Responses" (2007) 12 *First Monday* 6.

[72] See also Federal Trade Commission, "To Promote Innovation: the Proper Balance of Competition and Patent Law and Policy. A Report by the Federal Trade Commission," October 2003, Ch 3-V.

[73] *Ibid*.

[74] United States Patent and Trademark Office, 2007-2012 Strategic Plan. Available at: http://www.uspto.gov/web/offices/com/strat2007 (accessed 25 Feb 2009).

[75] See T Holbrook, "Return of the Supreme Court to Patents" (2007) 1 *Akron Intellectual Property Journal* 2.

[76] See *eBay v MercExchange* 126 S.Ct 1836 (2006).

[77] After the eBay decision, in fact, injunctions have been denied in cases where the patent holders did not manufacture or market the invention, or where the patent holder had already licenced the invention

Supreme Court to reduce patent thickets by affording more power to the examiners to reject applications on grounds of obviousness. *In re Comiskey*,[79] a decision on a software-related invention passed down by the Court of Appeal, Federal Circuit, highlighted and further discussed the *KSR*'s principles on the application of the non-obviousness requirement.

The discussion over patentable subject-matter has also been actively reintroduced in the discourse. Although the interpretation of the excluded categories from patent law has been the subject of debates for scholars, legislators and courts for a long time, this issue appeared to have been settled in 1998 with a Court of Appeal, Federal Circuit's landmark decision in the *State Street Bank* case.[80] By affirming that even though "mathematical algorithms are not patentable to the extent that they are merely abstract ideas," when such algorithms produce a "useful, tangible and concrete result" no reason should impede them to be considered patentable subject-matter. In fact, State Street officially opened the doors to both software and business methods patents in the US.

However, despite the fact that *State Street* has been a major cause of the increase in the patents filed and issued, as well as of the patent quality concerns in the fields of software and business methods, the Federal Circuit has now put forward a new line of interpretation, based on a more restrictive approach. The recent *In Re Bilski en banc* decision[81] and its "machine-or-transformation" test, represent a clear attempt to pose limits to the patentability of software and business methods inventions.

3.2 European Developments: What Europe Is Doing Wrong

While the American legal community has evidently started to realise the negative effects of software patent thickets and is embracing a more restrictive approach accordingly, the recent trends in European legislation as well as the EPO case law seem, instead, to be following a different track – with the EPO surprisingly lowering the bar for the issuing of software-related patents.

On the legislative side, harmonisation projects of patent laws – such as the COMPAT and the EPLA, as they currently stand – pose serious concerns towards the potential increase of patent thickets in Europe. On the one hand, the creation of a unitary litigation system is highly desirable in Europe. Under the existing rules of national enforcement, in fact, litigation costs are often very prohibitive. Additionally, this situation creates fragmentation, distortion of the internal market and uncertainty, making it very challenging for companies, users, courts and legislators to operate in the system.[82] On the other hand, the creation of a unique European patent court to deal

---

to others. See, for instance, *z4 Technologies, Inc. v Microscop Corp*, 434 F. Supp. 2d 437, 440 (E.D. Tex. 2006); *Paice LLC v Toyota Motor Corp*, 2006 WL 2385139, 5 (E.D. Tex. Aug. 16, 2006).

[78] *KSR Intern. Co v Teleflex Inc*, 127 S.Ct. 1727, 1739 ff. (US 2007).

[79] *In re Comiskey*, Fed. Cir. 2007-1286.

[80] *State Street Bank & Trust Co. v Signature Fin. Group Inc*., 149 F.3d 1368, 47 U.S.P.Q.2d 1596 (Fed. Cir. 1998). See also AT&*T Corp. v Excel Communications, Inc*., 172, F.3d 1352 (Fed. Cir. 1999).

[81] See note 34.

[82] For more information see R Ballardini, "Software Patents in Europe: the Technical Requirement Dilemma" (2008) 3 *Journal of Intellectual Property Law & Practice* 9, at 563-575.

with patent issues all across Europe raises patent "quality" concerns, as well as anxieties over the effectiveness of the future patent examination system. This last aspect, in particular, stresses the need to find a workable solution to patent thickets, in particular in the ICT field, where the problem reaches its peak. Neglecting this important step could lead to wrong policy decisions as well as inefficient solutions.

An example is the position recently supported by some of the major European ICT companies, such as Nokia, to oppose important harmonisation processes like the COMPACT and the EPLA, as they fear that the harmonisation on the litigation side would inevitably degenerate into a "charter for trolls."[83] Additionally, the unusual proposal put forward by IBM – to eliminate injunctions on patents that would be issued at European Community level on the framework of the Soft-IP project[84] – reflects this perspective.

However, it appears clear that the currently inefficient European litigation system would strongly benefit from harmonisation. It is also evident that harmonisation is not the cause of, nor would function as a direct aggravation to, the patent thickets or patent trolls in Europe. Instead, what this analysis most certainly highlights is that software patent thickets have their roots in very different reasons that thus need to be addressed and solved in the first place in order to create the proper basis for a harmonised European litigation system to function properly.

On the examination front, the most recent EPO case law clearly shows how the requirements needed to grant computer-implemented inventions patents are being increasingly relaxed.

It should be noted that the fact that computer programmes – "as such" – are excluded under Article 52(2)(3) EPC, does not mean that software-related patents are never granted in Europe. Specifically, the EPO grants patents on CII as far as they have technical character, are new and involve an inventive technical contribution to the prior art.[85]

The "any hardware" approach nowadays followed at the EPO,[86] however, radically limits the "as such" exclusion of computer programmes from European patent law. On the one hand, under the "any hardware" approach it is easy enough to satisfy the "invention = technical" test – it being sufficient to refer to some features of the hardware in the patent claims. The technical nature of the invention is, instead, assessed while examining the inventive step. In other words, under this practice, the non-patentable method (i.e. the computer programme) should be blocked at the inventive step examination, as no unobvious technical solution is provided to solve a technical problem. On the other hand, however, some recent case law[87] shows that

---

[83] See, for instance, E Hakoranta (Director, IPR Legal, Nokia Oyj), "Market for Patents. The Future of Fragmented IP & Boundaries of Patent System Explored" (2008), Current Trends in Patent Law Seminar, IPR University Center.

[84] J Sage, "Soft IP", Computer-Implemented Inventions: Where Do We Stand in the Debate on "Software Patents"?, Brussels 5

[85] See note 61.

[86] T 0931/195 *Controlling Pension Benefit System/PBS Partnership* [2001] OJEPO 441 and T 0258/03 *Auction Method / Hitachi* [2004] OJEPO 575.

[87] T0424/03 *Microsoft / Data transfer expanded clipboard formats* [2006].

this latter requirement might also be overlooked by the EPO. In this way, the seemingly absolutist "as such" exclusion appears to have lost any substantial meaning under current rules. The overall general increase in the number of software-related patents issued by the Office during the past ten years is the direct consequence of this trend.

Overall, this analysis suggests that, even though software patent thicket problems might still be relatively low in Europe if compared to the US, these concerns are likely to increase if immediate actions are not taken both on the legislative and jurisprudential side.

4. How To Improve The System?

From a theoretical point of view there are different solutions that might be adopted to clear the software patent thicket. Scholars, courts and legislators dealing with the issue have so far mostly concentrated on two of these types: eliminating patent protection for software,[88] on the one hand, and curtailing the patentable subject-matters[89] on the other.

Some have also focused on the proper application of the patentability criteria as a "passing filter" for valid patents. [90]Even in this latter case, however, attention has mostly been posed on the novelty and non-obviousness requirements. Disclosure matters, instead, have not yet been sufficiently considered. As a consequence, disclosure obligations remain remarkably low under current rules.

4.1 The Story Thus Far

As already mentioned, one of the proposals put forward in order to solve software patent thickets is the complete elimination of patent protection. In fact, ever since software has been considered patentable subject-matter, some supporters of the doctrine have been lobbying against the affording of a twenty year monopoly to such a dynamic, fast-changing field of technology. To give fuel to their arguments, scholars in various disciplines have also tried unsuccessfully to find a clear-cut answer to the question of whether patents promote or hinder innovation in the software field, in order to decide whether software should or should not be protected by patent law. Thus far, however, incentive theories in specific technological fields have been very difficult to support with empirical evidence. The incentive question, in fact, appears to be a matter of patent law in general, rather than of specific individual sectors of technology.

These questions still remain extremely challenging and finding an answer goes beyond the scope of this paper. Nevertheless, there seems to be strong evidence that banning patents for software would most probably be unrealistic: patent protection for computer programme applications exists and is clearly here to stay. Thus, it needs to be dealt with as an imperfect system that exists today.

---

[88] R Stallman, "The Danger of Software Patents" (2005). Available at: http://notabug.com/2002/rms-essays.pdf (accessed 25 Feb 2009).

[89] For example, *In Re Bilski,* note 34.

[90] See note 64.

Questions on software patentability under the "patentable subject-matter" doctrine have also been a matter of discussion among scholars, patent officers, legislators and judges for a long time. The impossibility of limiting the type of admitted claims in software-related inventions, however, is eloquently highlighted by the long European experience. In fact, the exclusion of computer programmes "as such" from the patentability field has taken the EPO on a crusade of trying to draw up the boundary between patentable and non-patentable objects in software. As already mentioned, the interpretation of the "as such" exclusion revolves around the "technical" criterion. That is: to be patentable subject-matter, inventions must be technical in nature. The difficulty of applying such a doctrine in software, however, has brought both the EPO and some national courts to embrace many different and inconsistent approaches over the years – leading to a general lack of legal certainty in the field.[91] This has also recently led to a referral to the Enlarged Board of Appeal in order to shed some light on the matter.[92] The same inconsistency also seeps out from the American jurisprudence (that precedes the full extension of patent protection to software and business methods inventions under the *State Street* doctrine[93]) and, more recently, from the difficult application of the rather clumsy "machine or transformation" test (from the Bilski case).[94]

An interesting, attempted, solution in the context of patentable subject-matter had also been put forward by the British courts, following the *Aerotel/Macrossan* decision[95] in 2006. This approach was based on the application of a test composed by four steps: first, to properly construe the claim; second, to identify the actual contribution in light of the prior art; third, to ask whether the actual contribution falls solely within the excluded subject-matters; and fourth, to check whether the actual or alleged contribution is technical in nature.

The application of the four-step test has certainly proven successful in reducing the number of CII patents issued in the UK.[96] From this perspective, therefore, the *Macrossan* approach could represent a great instrument for smoothing the patent thicket. Such a practice, however, might not be appropriate from a legal point of view. In Particular, the fact that the *Macrossan* approach goes back to the "contribution approach" that originated at the EPO from the *Vicom* decision[97] is a major point of

---

[91] See note 82.

[92] See Referral under 112(1)b) EPC by the President of the EPO (Patentability of programs for computers) to the Enlarged Board of Appeal, pending under Ref. N° G3/08 (23 Oct 2008).

[93] *Diamond v Diehr*, note 29 above; *Parker v Flook*, 437 U.S. 584 (1978); *Gottschalk v Benson*, 409 U.S. 63, 67 (1972).

[94] For more details see K Collins, "An Initial Comment on In Re Bilski: Tangibility Gone Meta" (2008). Available at: http://www.patentlyo.com/patent/law/collinsmetabilski.pdf (accessed 25 Feb 2009).

[95] English Court of Appeal, *Aerotel Ltd. v Telco and on the matter of patent application* GB 0314464.9, [2006] EWCA Civ 1371.

[96] See D Bainbridge, "Court of Appeal Parts Company with the EPO on Software Patents" (2007) 23 *Computer Law and Security Report* 2, at 199-204.

[97] See note 30.

controversy.[98] In fact, under this approach the examiner should decide whether there is an inventive step in order to ascertain the existence of an invention. However, under the EPC rules, access to patentability first requires an invention, and only then do novelty, inventive step, and industrial applicability need to be checked. This being so, the legitimacy of the contribution and, thus, also of the *Macrossan* approach, seems somehow lacking.

None of the attempted solutions mentioned have managed to provide a satisfactory answer to the software patent thicket issue which, instead, is still deeply entrenched into the system. What is certain, however, is that the transaction costs involved with the continuous formulation of proposals for new ways of tackling the problem do not only increasingly shift resources from innovation to litigation, but also undermine the purposes for which legal incentives are initially provided.

On the basis of this, this article suggests that the problem related to the software patent thicket should be addressed from a different angle. Specifically, the paper argues that patent law already possesses the appropriate instrument to address the issue, namely the requirement of sufficient disclosure in Europe and the enablement requirement in the United States.[99] A reform to improve the notice quality of issued claims should, thus, primarily come from the Patent Offices. Accordingly, examiners should ask for more information about the meaning of the claims, while rejecting vague and abstract claims more aggressively.

A more extensive disclosure could be a very effective tool to inhibit the growth of the software patent thicket, because it would reduce one of its major causes, i.e. the abstract nature of patent claims. Furthermore, when combined with a good prior art repository, this would support patent officers in conducting a more efficient quality examination, also in terms of inventiveness.

A comprehensive solution should also look at other fronts. For instance, the use of software patents in a more "ethical" way should be sought by the courts through the regulation of the use of injunctive relief. However, since the goal of the present article is to analyse the origin of the software patent thicket, and to propose solutions accordingly, these subsequently-arisen aspects of the problem are left for further investigation.

4.2 The Way Forward

4.2.1 A Better Prior Art Repository

As explained before, one of the major reasons behind the overlapping of IP rights in software is the absence of good prior art repositories. This has led to many non-novel and obvious patents being issued. Various initiatives have taken place recently on both sides of the Atlantic in this respect.

---

[98] For more details see note 82 above. See also W Cook, and G Lees, "Test Clarified for UK Software and Business Method Patents: But What About the EPO?" (2007) 29 *European Intellectual Property Review* 3, at 115-118.

[99] It should be noted that the recent Federal Circuit case law addressing the issue has focused in particular on the written description doctrine. See, for instance, *LizardTech, Inc. v Earth Resource Mapping, Inc.*, 424 F.3d 1336 (Fed. Cir. 2005). See also USPTO Written Description Training Material (25 Mar 2008).

The USPTO Strategic Plan 2007-2012 brought about the emergence of different programmes, mostly led by the open source community and aiming at improving the prior art repository in the field of software-related patents, in an effort to increase collaboration between the patent office, the scientific community and the industry. For instance, the Patent Commons,[100] the Peer to Patent,[101] the Open Source As Prior Art (OSAPA)[102] are some of the most well-known projects. In Europe some proposals have also been put forward, although on a more limited scale. The PatExpert project[103] is one of such examples.

These are all interesting and highly welcome efforts that should be encouraged and that will hopefully lead to more clarity and certainty in the assessment of the novelty and inventive step requirements of software-related inventions. However, it appears clear, not only that much more work needs to be done in this direction (in this respect, both private and governmental projects to clear the registries should be supported) but also that, even though these projects might smooth overlapping problems, they will most likely not entirely solve them. In fact, although searching problems might arise due to the complexity and vast number of patents in the field, absent other additional circumstances, a radical increase in resources and efforts to improve the prior art would, in the long run, solve the matter. In the context of software, however, it is much more complicated.

As extensively explained, the major reason for the growth of overlapping problems is the abstract nature of software patent claims, which, on the one hand, makes it easier for patent drafters to "play around" patent applications; yet on the other hand, it poses big challenges for the drawing-up of the novelty and inventiveness boundaries with respect to the prior art. This is mainly caused by the fact that in software patents the disclosure requirement is particularly weak.

Efforts should thus be addressed, not only at improving the prior art repositories, but also, and most importantly, at reducing abstraction concerns. To this end, the disclosure requirement should be enhanced.

4.2.2 More Disclosure Is Needed

The complexity of contemporary software inventions has led programmers to increasingly rely on high-level programming languages, hiding low-level details. As a result, software engineering has evolved towards higher levels of abstraction, jumping

---

[100] Patent Common project: http://www.patentcommons.org/ (accessed 25 Feb 2009).

[101] Open Source As Prior Art (OSAPA) project: https://www.linux-foundation.org/en/Osapa (accessed 25 Feb 2009).

[102] Prior aRT and Software Patents project: http://wiki.mozilla.org/Legal:Prior_Art#Summary (accessed 25 Feb 2009). Additionally, important repositories for open source projects in the field are Sourceforge: http://sourceforge.net (accessed 25 Feb 2009) and Freshmeat: http://freshmeat.net/about (accessed 25 Feb 2009). See also Open-IP: http://www.open-ip.org (accessed 25 Feb 2009).

[103] PatExpert: http://www.patexpert.org (accessed 25 Feb 2009).

from simple routines and procedures to more organised units like modules, packages, object classes and components.[104]

Instead of following such an evolution, however, the level of disclosure required in patent applications has remained unchanged, resulting in a generally inadequate instrument to unveil software-related inventions in patents. This is particularly detrimental for the software sector due to the fact that the construction of the claims tends to be much more ambiguous in such abstract technologies and, thus, more extensive information and examples are usually necessary in order to understand and reproduce the invention.

It remains clear that different types of software inventions call for different levels of disclosure. Hence, the optimal level of disclosure in a software patent can vary from a high-level architectural description, to flowcharts,[105] pseudocodes,[106] or source codes.[107] In order to increase patent notice and, in turn, reduce abstraction in software-related patents, none of the mentioned forms of expression should be exclusive. Instead, they should complement each other. This being so, however, at least flowcharts or pseudocodes should always be required in the applications, along with the description of the invention in natural language. In fact, although, in general, flowcharts might be a more workable tool, it is not difficult to imagine situations where flowcharts or diagrams would not be sufficient to show that the applicant is in actual possession of the claimed invention. For example, the path along an arrow from A to B might actually be impossible to practice for a person skilled in the art without the associated pseudocode.[108]

Additionally, when necessary, the source code might also be requested. In these cases the submission in a format like CD-ROM, or other mass storage devices that may be used by examiners to inspect efficiently and effectively the computer programme code listings, would be more suitable than submission in paper. Specifically, as source codes could be considered the "genes" of software, a disclosure policy could be built on the lines of the escrow of biological samples in biotechnology patents. In the same way as for biotech patents, providing the source codes might be required when the invention cannot be described in a patent application in such a manner as to enable it to be reproduced by a person skilled in the art. [109] Moreover, in the same way as for

---

[104] B Hall, "On Copyright and Patent Protection for Software and Databases: A Tale of Two Worlds", in O Granstrand (eds) *Economics, Law and Intellectual Property. Seeking Strategies for Research and Teaching in a Developing Field*, Ch 11 (Kluwer Academic Publishers 2004).

[105] A flow chart (or flow diagram, or flow sheet) is a schematic representation of a sequence of operations, as in a manufacturing process, or computer programme.

[106] Pseudocode is a form of 'structured English' that looks like the source code of a high level programming language, but is more generic and somewhat more easily readable. Pseudocodes are easier for humans to understand than conventional programming language code, as they typically omit details that are not essential for the human understanding of the algorithm.

[107] Source code refers to any sequence of statements of declarations written in some human-readable computer programming language. Source code allows the programmer to communicate with the computer using a reserved number of instructions.

[108] See note 104.

[109] See Implementing Regulations to the Convention on the Grant of European Patents, Part II-5, Rule 31-32; Directive 98/44/EC of the European Parliament and the Council of 6 July 1998 on the legal protection of biotechnological inventions, Article 13; 37 Code of Federal Regulations (CFR), s 1.808.

biotech patents, access to the deposited material (i.e. the source code) should be restricted until the issuing of the patent, as well as in cases of refusal or withdrawal, in order for the applicant not to lose important trade secrets.[110]

As already mentioned, this policy would find justification in the sufficiency of disclosure (in Europe) and the enablement requirement (in the US) of patent law.[111]

4.2.3 Enhancing Disclosure To Smooth The Thicket

There is reason to believe that requiring software patent applicants to provide flowcharts, pseudocodes or, when necessary, source codes, in addition to the clear description of the invention in natural language, would help in affording a more adequate, less broad, level of protection by reducing one of the major causes of the overlapping problems in software-related patents – that is, abstraction. The enhanced notice would limit the number of patents filed, which would thus also indirectly smooth the thicket.

Generally speaking, the patent system is structured to encourage patent filing early in an invention's development.[112] When applying for a patent in the software field, for instance, one does not necessarily need to have produced any flowchart, pseudocode, nor line of source code. Requiring applicants to provide this information and, in so doing, pushing them to file at a later stage in their idea's conceptualisation, would lead both to less abstract applications and to a reduced number of patents filed. Flowcharts, pseudocodes, or source codes would constitute an additional, better "proof" that the applicant has an invention and, in this way, would slow down the filing of very uncertain applications. Furthermore, it would also make it easier to tighten patent protection to the specific function produced by the mentioned design documents or lines of code. In turn, this would lessen the abstract nature of the claims.[113]

Postponing the applicants' filing time would be possible in the software field in particular because the R&D costs that inventors have to bear in order to create further innovation (thus, the monetary incentives needed) are relatively low.[114] The cost of R&D in software has been made even smaller by current technology that adopts automated tools, for example, to generate sections of codes so as to help designing simple programmes like websites. Moreover, when products are intangible information, as in the case of software, the cost and speed of imitation (as well as the marginal costs of production) is also usually low. In fact, while writing a programme takes relatively little time and money, the debugging of such a programme is still a

---

See also Budapest Treaty on the International Recognition of the Deposit of Microorganisms for the Purposes of Patent Procedure, 28 Apr 1977.

[110] *Ibid.*

[111] See EPC, Article 83 and 35 U.S.C. § 112.

[112] See J Duffy, "Rethinking the Prospect Theory of Patents" (2004) 71 *University of Chicago Law Review* 2, at 439-510.

[113] See K Rowe, "Why Pay for What's Free?: Minimizing the Patent Threat to Free and Open Source Software" (2008) 7 *John Marshall Review of Intellectual Property Law* 3, at 595-620.

[114] D Burk, and M Lemley, "Policy Levers in Patent Law" (2003) 89 *Virginia Law Review* 7, at 1575-1696.

significant undertaking.[115] With software reverse engineering being an inefficient tool for revealing patented software-related inventions and the source code often being kept secret, copying is usually quite difficult.

Various studies on innovation, in fact, have documented the weakness of using patents for protecting software-related inventions.[116] Companies mostly acquire software patents not to protect their inventions, but rather for "strategic" reasons, such as to support complex cross-licensing agreements, or in response to lawsuits alleging infringement, or as "patent signals" to show to their potential partners that they have a powerful research and development arm, or, again, to increase the company's value.[117]

Overall, this analysis suggests that increased disclosure would not reduce innovation incentives in software. On the contrary, it would greatly benefit both the patent examiners and the third parties in understanding the patented inventions and, in so doing, it would provide more incentives for further progress. Major reasons of concern, in fact, are that software-related inventions do not transpire "from the face" of the patents – they are very obscure and not readily understandable and, consequently, difficult to be built upon. This is also one of the justifications of why software-related patents are mostly ignored by researchers and companies.[118]

Reducing the abstract nature of the claims would also help to improve analysing applications in terms of prior art. As seen, the abstract nature of the software patent claims makes it difficult for patent examiners to draw the boundary between allegedly new, inventive inventions and prior art: comparing abstract concepts so as to find relevant pieces of prior art is a remarkably challenging task. Lowering the level of abstraction would, to a certain extent, also reduce this problem.

Although this type of policy would mostly address future applications, a better disclosure would also provide great support for the earlier mentioned projects[119] that aim at improving the prior art repositories and clearing the "bad" patents issued.

Another advantage, deriving from a flowchart-code disclosure policy, relates to the breadth of the patent scope. In incremental industries, as is the case with software, in order to maximise progress, the patent scope should be kept limited and patents should not extend to several product generations: this would, instead, stifle innovation in subsequent incremental developments – representing a huge hindrance to

---

[115] *Ibid*.

[116] *Ibid*. See also W Cohen, R Nelson, and J Walsh, "Protecting Their Intellectual Assets: Appropriability Conditions and Why U.S. Manufactory Firms Patent (or Not)" (2000). NBER Working Paper Series No 7552, available at: http://www.nber.org/papers/w7552 (accessed 25 Feb 2009); A Arundel, "The Relative Effectiveness of Patents and Secrecy for Appropriation," (2001) 30 *Research Policy* 4, at 611-624.

[117] See, for instance, C Long, "Patent Signals" (2002) 69 *University of Chicago Law Review* 2, at 625-680. See also R Merges, "Software and Patent Scope: A Report From The Middle Innings" (2007) 85 *Texas Law Review* 7, at 1627-1676.

[118] See M Lemley, "Ignoring Patents" (3 Jul 2007). Stanford Public Law Working Paper No. 999961. Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=999961 (accessed 25 Feb 2009). See also R Mann, "Do Patents Facilitate Financing in the Software Industry?" (2005) 83 *Texas Law Review* 4, 961-1030.

[119] See notes 100-103.

technological progress.[120] Increasing disclosure would also help in meeting these goals: as the scope of protection in patent law is defined by the claims,[121] requiring applicants to provide more information on their inventions would directly result in a reduction on the scope of the patents as the enhanced notice would limit the claims.

Additionally, providing greater information would support judges dealing with infringement cases in delimiting the scope of protection of the challenged patents. In fact, although it might be easy for someone skilled in computing to understand whether there is the same invention (i.e. the same function) behind two allegedly different software-related patents (by using, for example, reverse engineering techniques) it may be unclear in the eyes of non-technically trained judges. Thus, more detailed information, as well as different tools, might be necessary in order to convince a court that certain software is or is not infringing upon a patented one.

It is important to stress that this article does not suggest requiring applicants to file only once a complete product has been developed (e.g. when the entire source code is available). Although such a condition would most likely fulfil the disclosure requirement in the majority of cases, it would clearly discriminate patentees in the software arena due to the great deal of work experimentation that would be necessary before the filing of an application. In this way, a mandatory source code requirement would constitute a degree of disclosure that would go far beyond the norm for the other kinds of patents, and could thus raise legitimacy concerns under Art. 27 and 30 of the TRIPs Agreement,[122] which prohibits EU member states from discriminating on the granting of patents based on the type of technology at issue.[123] A full source code disclosure, in fact, would require that an invention be disclosed so that, not only a skilled programmer, but also a person of virtually no programming experience would be able to make it and use it. Such a requisite would also most likely be highly inefficient from a practical point of view: providing too long a description in a particular programming language could, in fact, make the disclosure even worse – drowning the important facts into the unimportant ones. Additionally, it could create searching problems, delaying filing, as well as the examining and issuing of patents in the field. Overall, this could degenerate into either too broad or too narrow patents being issued.

What this article proposes, instead, is to improve disclosure functions by affording more power to examiners that allows them to reject very abstract and unclear applications, and, especially, by pushing applicants to always use flowcharts or pseudocodes along with a clear description of the invention in natural language. Computer programme code listings, instead, should be requested only in the case and to the extent that they are necessary to enable the skilled man. This would burden applicants or examiners in the software field no more than in any other field of technology.

---

[120] See note 109.

[121] EPC, Article 69; 35 U.S.C. § 112.

[122] WIPO Treaty on Trade Related Aspects of Intellectual Property Rights, Morocco, 15 Apr 1994 (TRIPs).

[123] See G Dinwoodie, and R Dreyfuss, "Diversifying Without Discriminating; Complying with the Mandates of the TRIPs Agreement" (2007) 13 *Michigan Telecommunications and Technology Law Review* 2, at 445-456.

4.2.4 Possible Drawbacks: Some Remarks On Trade Secrecy

Although increased notice in software patents applications would bring important advantages, it could also potentially raise some concerns. For instance, it could be argued that enhanced patent disclosure could lead to a more extensive use of trade secret protection. Overall, trade secrets might be even more detrimental than patents because information is kept undisclosed. Some characteristics of the software industry, however, suggest that such a threat is probably not very likely.

In the early days of software protection the use of trade secrets was thought to be problematic in the code context because of the ease with which the secret status could be lost.[124] At a time like the one in which we are living – where DRM mechanisms are becoming increasingly sophisticated and, in particular, where software developers usually do not disseminate source code without first encrypting it, using passwords, or restricting access to information placed on computers – trade secrecy has certainly become a more useful and appropriate instrument of protection for the software code. Nowadays, in fact, inventors simultaneously combine the use of trade secrets and patents, as well as licensing agreements and DRM to protect the functional aspects of their software.

For widely distributed commercial software, however, keeping information of the programme completely secret is a highly challenging task. This is made particularly difficult by the fact that outsourcing work overseas has become increasingly common.[125] Additionally, when the trade secrets are maintained through licencing contracts distributed with the software, enforcement is almost impractical as these contracts are constantly ignored. Trade secrecy might also lead to very inefficient practices, like keeping the entire code undisclosed. In component industries like computer software, though, the need to access sections of the codes so as to "fit the pieces together" might often be necessary.[126]

Finally, even if enhanced patent notice would lead to less information being disclosed, it is not immediately clear whether this would hurt innovation in the software field in the long run. Recent studies have demonstrated that researchers and companies in component industries mostly ignore patents.[127] Under current rules, in fact, software patents are so obscure that they are effectively secrets. This failure of software patents to pursue their traditional function of spreading technological knowledge indicates that most innovations in the field are "independent" inventions.

This not only supports the thesis that disclosure does not work particularly well in the software context but it also suggests that a potential increase in the use of secrecy would not reduce innovation in the field. On the contrary, if anything, it would probably curtail the dense web of patents by discouraging the filing of too abstract and uncertain applications.

---

[124] K Canfield, "The Disclosure of Source Code in Software Patents: Should Software Patents Be Open Source?" (2006) 7 *Columbia Science and Technology Law Review* 6.

[125] *Ibid*.

[126] *Ibid*.

[127] See note 114.

5. Conclusions

The problems related to software patent thickets have been openly recognised by scholars, courts and legislators. Accordingly, in recent years, various initiatives have been put forward to address the issue. Efforts, however, have thus far been focused merely in improving prior art repositories; curtailing the patentable subject-matters; increasing the non-obviousness bar; and reducing the use of injunctive relief. Issues related to the abstract nature of software have, on the contrary, not been properly assessed. However, this article demonstrates that abstraction is one of the major causes of the growing software patent thicket and, as such, represents a fundamental issue to be addressed.

Specifically, this article suggested that a more extensive disclosure, when combined with a good prior art repository, can be a potentially successful tool for reducing the effects of the abstract nature of software-related patent claims and, in turn, inhibit the patent thicket. To this end, inventors should be requested to include in their applications the flowcharts, pseudocodes or, when necessary, parts of the source codes along with a clear description of the invention in natural language.

Overall, this would not only serve patent officers in better evaluating the applications, but would also support judges dealing with questions of infringement. Competitors and new inventors – who necessarily need detailed information about the programmes behind the patented inventions in order to improve upon them – would also clearly benefit from such a practice.

ESSAY IV

Rosa Maria Ballardini

Proprietary Software vs FOSS: Challenges with *Hybrid* Protection Models

IPR University Center Publications, IPR Series B:4 (2012)

Peer reviewed

Proprietary Software vs FOSS: Challenges with *Hybrid* Protection Models

*Rosa Maria Ballardini**

Abstract

This article analyses the reasons and consequences of the fact that open source software has become a portion of the technology used by proprietary companies. It focuses on problems arising from the use of what is designated here as the 'hybrid' protection model by commercial companies. The term 'hybrid' model refers to a situation where companies incorporate both open source and proprietary code into the final software they release to the market. The coexistence of both the proprietary and the open source software model is essential to promote innovation in the software field. Due to the different and allegedly conflicting principles under which they are based, however, the relationship within the two systems might not always be peaceful. By combining legal theory and empirical research, this paper provides a comprehensive analysis of the "core" legal challenges surrounding the implementation of the 'hybrid' model in the context of commercial software, and sheds light on the coping mechanisms companies implement in order to navigate such risks.

* Researcher in IP law at HANKEN School of Economics; member of the INNOCENT Graduate School in IP law, IPR University Centre, University of Helsinki. The author thanks Professor Niklas Bruun, Professor Marcus Norrgård, Professor Graeme Dinwoodie, LLM Wim Helwegen, and LLM Tanja Liljeström for the valuable comments. Sincere thanks also go to all the companies and people who kindly contributed to the empirical part of this research.

1. INTRODUCTION

This article investigates how firms are developing the relationship between proprietary (mostly copyright and patents) and open source software (OSS/FOSS/FLOSS[1]) under current rules. Specifically, the paper analyses the reasons and consequences of the fact that open source software has become a portion of the technology used by proprietary companies.

In recent years, the proprietary and the FOSS protection models have been increasingly used simultaneously in the same software packages. In fact, almost every company operating in the software field uses both open source and proprietary software. One could argue that the coexistence of both the proprietary and the open source software model is essential to promoting innovation in the software field. Due to the different and allegedly conflicting principles under which they are based, however, the relationship within the two systems might not always be peaceful.

Thus far, research has focused on either FOSS as a phenomenon or proprietary software individually. Very few studies have investigated the relationship and interactions between the two. As the tie between open source and proprietary software models stretches, the need for an in-depth analysis of the ways open source software affects companies' IP policies (and vice versa) is becoming clear.

This article primarily aims at investigating problems arising from the use of what is designated here as the *hybrid* protection model by commercial companies. The term *hybrid* model refers to situations where companies incorporate both open source and IP protected code into the final proprietary software they release to the market. Both the respective advantages and disadvantages of proprietary and open source software are investigated. A practical analysis using company examples is conducted to expose some of the strategies that firms have used to mix open source and proprietary software features together. The analysis is based on a literature review and its intent is to shed light on the major legal challenges involved with *hybrid* models.

The second part of the paper composes of an empirical study in the form of a case study research. A case study analysis was chosen because an in-depth investigation was needed to provide a holistic understanding of the problems. To this end, the case study relied upon a qualitative-type of analysis. A quantitative technique would have probably obscured some of the important information that needed to be uncovered, such as whether the *hybrid* protection method is efficient and what kind of specific problems it involves in practice. The theoretical focus (i.e., the object) of the study was identified as the problem(s) encountered by commercial companies that implement *hybrid* models of protection for their software products. The subject of the study was portrayed by representative companies operating in

1 Open Source Software (OSS), Free and Open Source Software (FOSS), Free, Libre Open Source Software (FLOSS), are all slightly different alternatives used to describe software which can be used, modified and redistributed with little or no restrictions. For the purpose of this paper, however, these terms will be used interchangeably.

the software field. A multiple case study was conducted, as more than one case was available for replication. Several companies use the *hybrid* model and might encounter the problems identified in this study. The study relied upon two different sources of evidence: documents and interviews. The case study was relevant because it provided in-depth answers to the theoretical issues formulated in the first part of the paper. The empirical analysis shed light on the most concrete legal risks associated with *hybrid* models, and on the coping mechanisms used to navigate such challenges.

2. THE PROTECTION MECHANISMS

Proprietary and open source software models possess very different characteristics and work in very different ways. When both mechanisms are used in the same software package, the divergences can lead to unavoidable conflicts and legal risks. To understand where the potential controversies might arise, it is important to first understand the ways they function within this model.

2.1. The Proprietary Model

A major characteristic of proprietary software lies in the way that computer programs are developed. Specifically, the proprietary model favours a centralised, closed type of development where the product is fully "built in-house". As a consequence, the developing firm usually owns and retains all the rights over the software it produces.

Another important feature of the proprietary software model lies in the way software is distributed to users. Generally speaking, licensing is central to the exploitation of all types of intellectual property rights. In software licensing, the IP holders retain ownership, but grant the licensees rights to use the software subject to certain restrictions. The type of restraints placed on users differs between proprietary and open source software, and is one of the main points of contention in the closed-open software conflict.[2]

Proprietary software companies use various types of end user licensing agreements that provide the licensees with limited rights to use the software for specific purposes. For instance, both the copyright and the patent regimes tie the license price into the usage restrictions.[3]

Under the proprietary model (or "closed-code" model) IP owners do not make their source code available to end users, and the product is distributed only in object code form. The rationale behind this trend is that source code contains valuable trade secret information that cannot be protected under the copyright or the patent regimes, and therefore should be kept a secret. Neither of these two legal

2 D Evans, & A Layne-Farrar, "Software Patents and Open Source: the Battle Over Intellectual Property Rights" (2004), 9 *Virginia Journal of Law and Technology* 10.
3 M Välimäki, *The Rise of Open Source Licensing. A Challenge to the Use of Intellectual Property in the Software Industry* (2005), Ch. 2, at 13-49.

definitions provides protection to the structure, ideas, and logic described in the source code.[4] Therefore, it is important to point out some issues surrounding the debate on the treatment of source code.

First, it should be noted that even though patent rules do not prevent disclosure of the source code, they don't require it either. Additionally, the special nature of software, as well as existing case law and legal dispositions, have all contributed in drastically curtailing the disclosure requirements for software-related patents.[5] Consequently, it has been a common practice to disclose only the minimum amount of information necessary in order to meet patent law requirements.[6] Due to the cumulative nature of computer software and the way the code is constructed, however, access to the source code (or at least to detailed interface descriptions) is often necessary for developing new programs and fostering innovation. This need is particularly evident when (as it is often the case) the software product is a development tool or a component that needs integration adaptability.[7]

The "closeness" of the proprietary model distinguishes it from the open source model, and represents a highly controversial aspect in the debate between proprietary and open source advocates.

2.2. The Open Source Model

The most distinctive features of the open source system are identical to those that distinguish the proprietary model: the development structure, the licensing scheme and its relationship to intellectual property rights, and the policy treatment of the source code.

On the development side, a FOSS peculiarity is the use of collaborative developing structures that extend beyond the boundaries of a single firm. Open source projects are controlled by a community of stakeholders and the software is usually developed by a group of self-organised collaborators.[8]

Nowadays, the term "open source license" describes several different licenses. In order to be recognised as FOSS, licenses need certification from the Open Source Initiative, a non-profit organisation dedicated to promoting open source software.[9] The two most frequently used FOSS licenses are the GNU General Public License (GPL)[10] written in 1989 by Richard Stallman for use of programs released as part of the GNU Project, and the Berkeley Software Distribution (BSD) licence[11], forged by Bill

4 See note 2 above.
5 See D Burk, and M Lemley, "Designing Optimal Software Patents", in R Hahn (eds) *Intellectual Property Rights in Frontier Industries: Software and Biotechnology* (AEI Press 2005).
6 See EPC, Article 83, and 35 U.S.C. § 112.
7 RM Ballardini, "The Software Patent Thicket: A Matter of Disclosure", (2009) 6:2 *SCRIPT-ed* 207, http://www.law.ed.ac.uk/ahrc/script-ed/vol6-2/ballardini.asp.
8 See note 3 above.
9 See http://www.opensource.org/.
10 The version of the license currently in use is version 3, released in 2007. See GNU General Public License, Version 3 (2007) at: http://www.gnu.org/copyleft/gpl.html.
11 See http://www.opensource.org/licenses/bsd-license.php.

Joy from the University of California at Berkeley. The GPL and the BSD have served as models for many other FOSS licenses. In particular, it can be said that they gave birth to two highly influential families of FOSS licenses: the "copy-left" family, derived from the GPL license, and the "academic families", derived from the BSD license.[12] Each FOSS licence is based on copyright law. Specifically, FOSS licenses set the relationship between the copyright holder and the users.[13]

OSS licenses will be analysed in more details later in this paper, however, it is important to note that these licenses clearly differ from any traditional type of IP license. While IP licenses impose restrictions on the use of certain products, the OSS licenses grant freedom to use the licensed software. Specifically, OSS licenses grant the royalty-free right to run, modify, distribute, and redistribute modified versions of the computer program.[14] Another essential characteristic of OSS licenses is the obligation for the licensor to make the source code "freely" available to developers and users but this "zero-royalty" feature of OSS licenses does not necessarily mean that OSS licensors cannot profit from selling the code. OSS licenses are non-exclusive: copyright owners might license their code under an additional license that provides additional services, such as a warranty to the users.[15]

Notwithstanding the common characteristics highlighted above, FOSS licenses differ in several respects. One of the most important distinctions lies in the extent the license allows commercial exploitation, especially with respect to the possibility to combine the FOSS code with a company's proprietary code. The essential nature of the GPL, for instance, is enshrined in a requirement called "copyleft":

> "You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License".[16]

Thus, under the GPL, licensees might be required to disclose their own source code under an open source license. On the other hand, copyleft licenses are not uniform in this aspect. For example the Lesser General Public License (LGPL)[17] and the Mozilla Public License (MPL)[18] are more permissive than the GPL.

Finally, the BSD and academic licenses in general allow more commercial exploitation and more flexibility in combining open source with proprietary software.

It is important to keep all this in mind because (as will be explained later) these differences might be crucial for firms attempting to profit from FOSS products.

---

12 See note 3 above.
13 *Ibid*.
14 *Ibid*.
15 *Ibid*. See also F Lévêque and Y Ménière, "Copyright Versus Patents: the Open Source Software Legal Battle" (2007), 4 *Review of Economics Research on Copyright Issues* 1, at 27-46.
16 See GNU Lesser General Public License, version 2, at http://www.gnu.org/copyleft/lesser.html, "Terms and Conditions for Copying, Distribution, and Modification", 2b).
17 *Ibid*, GNU Lesser General Public License.
18 See Mozzilla Public License Version 1.1. at http://www.mozilla.org/MPL/MPL-1.1.html.

2.3. Closed vs. Open Source

This section compares the proprietary and open source software models to shed light on the benefits and limits each model brings to software innovation. Both systems possess advantages and disadvantages with respect to software innovation, and therefore necessitate finding a balance.

2.3.1. Advantages and Limits of Closed Software

Commercial developers and vendors typically protect their software using many different IP mechanisms, with copyright and patents being the most popular. The reason for this multilayered type of protection lies in the pluralistic nature of computer programs.

Computer programs possess several elements, each of which could fall into different categories of IP laws. Software has a dual nature: on the one hand, it can be defined as a literal work under the form of source code and object code. On the other hand, once executed by a machine it is also a functional object. This configuration allows software to fit within the scope of many different types of IP tools.

The major advantage of proprietary software lies in the fact that a firm can control the destiny of its products. This provides companies with the sufficient profit motive to make their products highly valued by consumers. For example, vendors might try to make their products backwards compatible with earlier versions, so that consumers can experience a smooth transition from an older version to a newer one.[19]

Another strength of proprietary software is that it provides consistent platforms for running applications. Fragmentation is usually not an issue for proprietary software. In an industry like software, where network effects are very strong, this consistency is especially important. For example, developers of proprietary software, such as Windows, write their programs in a manner that allows them to run on all computers meeting their specific hardware and operating system requirements (e.g. Windows ME, Windows 2007, etc…).[20]

Each individual IP protection mechanism possesses both positive and negative aspects with respect to software protection. As previously mentioned, the focus here is on copyright and patents.

For many years, software has been considered a literary work and primarily protected by copyright law.[21] It is undeniable that the software code is expressed "in writing". To some extent, copyright law provides computer code with an adequate level of protection. Copyright prevents third parties from copying the software code

---

19 See D Evans, and B Reddy, "Government Preferences for Promoting Open-Source Software: A Solution in Search of A Problem" (2003), 9 *Mich. Telecomm. Tech. L. Rev*. 313. Available at: http://www.mttlr.org/volnine/evans.pdf.

20 *Ibid*.

21 See 17 U.S.C. §102(a) (1988); Software Copyright Directive, Art. 1; see also TRIPs Agreement, Art. 10.

(binary and source code) without permission, or using it as an input into a product of their own. On the other hand, however, copyright presents various shortcomings when applied to software. In this respect, the major reasons of concern lay on the copyright scope of protection.

Probably the biggest failure of software copyright is that copyright law extends to the 'expression' of the program in the form of either source or object code, but does not afford protection to the way the program works, i.e. to the program's functions ("idea-expression" dichotomy). On the one hand, by leaving the functional elements unprotected, copyright creates serious risks of under protecting software. On the other hand, there are risks that over-protection might arise, as drawing a border between the literal and functional elements of software is often difficult. This is especially due to the fact that in computer programs there is a significant degree of independence between literal and functional manifestations. The software functions are fully independent from the grammatical (i.e. literal) construction of the lines of code. In other words, even though the source codes of two programs might look completely different, such codes can perform the exact same function and produce the same (or a very similar) set of instructions. This configuration makes it difficult to discern the literal/expressive from the functional/utilitarian in software.[22] Additionally, other problems might arise with software copyright. These can include the long duration of copyright protection with respect to the software's short lifespan, the challenges involved with the definition of "originality" of the code, and various other difficulties deriving from new technological developments (for example, the increased use of modularisation and object-oriented designs in computer programming).[23]

Copyright shortcomings are partially responsible for the adoption of patent protection for computer-related inventions. Historically patent protection was not available for software based on the perception that computer programs were abstract concepts, and as such did not meet general patentability requirements. Current doctrine, however, recognises the patentability of computer software (in particular in the United States and, to some extent, in Europe).[24] By impeding competitors from writing code that includes any patented aspect of the software, patents constitute an efficient mechanism for blocking, or at least obstructing, attempts to duplicate a program's functionality.[25] Patents can protect the implementation of algorithms and other creative aspects of software design.

Notwithstanding these advantages, software-related patents possess various shortcomings. The fact that assessing patentability of abstract technologies such as computer programs raises quality concerns, and potentially leads to issuing of obvious, non-inventive patents in the industry. This can create patent floods, hold up

22 For a general discussion on the issue see RM Ballardini, "Scope of IP Protection for the Functional Elements of Software", in *In Search of New IP Regimes* (2010), Publications of IPR University Center, at 27-62. Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1599607.

23 See J Lipton, "IP's Problem Child: Shifting the Paradigms for Software Protection", (2006) 58 *Hasting Law Journal* 2, at 205-251.

24 For more information see note 7 above.

25 R Jordan, "On the Scope of Protection for Computer Programs under Copyright Computer Programs: The Patent/Copyright Interface", (1989) 17 *A.I.P.L.A. Q.J.* 3, at 199-214.

problems, royalty stacking, and blocking patents. It might actually discourage rather than promote innovation in the field.[26]

Another possible disadvantage of the proprietary software model refers to the treatment of the source code. Specifically, the fact that the source code is kept undisclosed does not allow a technically adept user to fix bugs himself or customize a proprietary program in ways the vendor has chosen to make the program available. In this way, the non-disclosure of the source code might impede direct and indirect competitors, as well as end users, to build upon the program in order to create further developments. Even though these activities do not usually affect the vast majority of home users they might be very important for large customers, and in particular, for promoting further software innovation.[27]

To summarise, the extent of intellectual property protection that computer software should receive is debatable. Proprietary models present various advantages, but also several disadvantages with respect to software innovation. At the same time, however, it is untenable to argue that no IP protection should be available for software.

2.3.2. Advantages and Limits of Open Source Software

Depending on the product, its usage, and the market constraints, OSS has specific properties that can be advantageous or disadvantageous for computer software.

The most remarkable strength of the FOSS model is costs saving. Using FOSS can save both license and development costs. Furthermore, it can save time for the component updates and corrections because more free labour is available to localize and correct defects. It should be pointed out, however, that open source software is not free software, and it often requires substantial investment in order to deploy it in the marketplace.[28]

Another advantage of widely used OSS packages is the generally high quality of the software. The fact that there is a large community behind the projects guarantees that bugs are found and fixed quickly.[29] In this sense, another important reason for using OSS is to attract developers to reduce costs related to having internal support services.

The fact that the source code is made available to developers and users is another remarkable advantage of open source software. Openness can enhance welfare in several ways: it allows others to correct defects and bugs and to customize the programs by adding more features to the software, designing around it, and

---

26 See M Lemley, and C. Shapiro, "Patent Holdup and Royalty Stacking" (2007) 85 *Texas L. Rev.* 7, at 1991-2050.

27 See note 19 above.

28 M Ruffin, & C Ebert, "Using Open Source Software in Product Development: A Primer", 21 *IEEE Software* 1 (2004).

29 See "Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defence" by *MITRE Corporations*, Version 1.2.04 (2003).

promoting further developments. Furthermore, the openness of the code makes it easier to adapt and reuse it, and ultimately helps software retain value rather than becoming obsolete.[30]

On the other hand, the fact that the code is open can also bring some unavoidable disadvantages, as developers have limited opportunities to earn monetary returns for their investments. Even though non-pecuniary rewards might provide some motivation[31], the limited economic benefits of OSS can reduce the supply of efforts devoted to these activities.[32] For instance, the limited pecuniary rewards in open source projects might reduce firms' incentives to perform costly consumer research into usability and consumer needs.[33]

Furthermore, it is worth noticing that proprietary companies tend to avoid digging into the source code of large OSS projects unless absolutely necessary. Widely used OSS packages usually include very large source codes, therefore making it difficult to try to change parts of such code (especially if the programmer is not very familiar with that particular software). For example, in dealing with bugs in the code most proprietary companies would look for fixes (e.g. whether there is a newer version of the software that has been fixed), or check whether the bug has been reported (and in this way they might find out whether the OSS community is already working on fixing the bug, or whether a fix has been scheduled for future releases). If these options are not available they might decide to find a workaround, which might not mean fixing the bug but rather not using the functions of the software that have the problem. There are cases where the openness of the code is extremely useful. For example, when a company wants to make the binary smaller by taking only part of the source code, or when a company needs to customize the software for internal use. In a nutshell, the fact that the code is open is definitely a positive characteristic of open source software, but it is not the most important one of the model.[34]

From a company perspective, another advantage of OSS is the recruiting process. The fact that extensively used OSS projects develop large communities means that there are many developers familiar with using the software tools related to such projects. Thus, incorporating widely used OSS projects can increase the overall efficiency of the company. If the recruited developers are familiar with the OSS software the company is implementing, they will certainly be more productive.[35]

Theoretically, a fundamental problem with the OSS model is market fragmentation due to the development of multiple, sometimes incompatible, versions of the same software. The extent to which open source users take advantage of their freedom to

30 S Maurer, and S Scotchmer, "Open Source Software: the New Intellectual Property Paradigm" (2006), *National bureau of Economic Research*.
31 See, for instance, K Lakhani, & R Wolf, "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects" (2005) in J Feller, B Fitzgerald, S Hissam, & K Lakhani, *Perspective of Free and Open Source*, MIT Press, Mass.
32 See note 19 above.
33 *Ibid*.
34 Interview conducted on 12 October 2011.
35 *Ibid.*

modify and customize code leads to fragmentation.[36] In practice, however, when OSS is a *de facto* standard component and has a large user community, it provides lasting solutions. The advantage is that developing an application on a proven *de facto* OSS standard provides companies with better protection against changes in a supplier's terms or conditions. With a proprietary solution, changing suppliers is often not possible because the mitigation costs are usually too high.[37]

3. THE RAISE OF THE *HYBRID* MODEL

The above analysis suggests that both open source and proprietary mechanisms are important to promote innovation in the software field. Indeed, the analysis exemplifies not only that each model possesses both positive and negative aspects, but also that the different and often conflicting aims of the proprietary and open source systems can hamper their peaceful coexistence when both models are embedded into a company's final proprietary software product. Discrepancies arise from their different perspectives, spanning from managerial, economic, and technology-related, to purely legal matters.

This article investigates what is here termed the *hybrid* protection model for computer software. In the *hybrid* model companies incorporate both open source and proprietary code into the proprietary software they release to the market. Specifically, the *hybrid* model takes advantage of the resources available through FOSS while adding proprietary features that generate revenue. Potentially, this could provide equivalent value to traditional commercial software at a lower cost and with better quality to the end users.[38]

Specifically, the focus of this paper lies on the legal challenges that can arise from the *hybrid* model and the possible coping mechanisms that can help navigate the risks.

The *hybrid* model includes different companies' trends and strategies. On the one hand, some typical proprietary companies are increasingly including OSS code into their proprietary software. On the other, traditional OSS companies are starting to incorporate intellectual property rights into their model. Furthermore, other companies were born as "pure hybrids" in the sense that they have included both OSS and proprietary features into their models from their beginnings.

Corporations have launched various strategies as part of these transformations. The following paragraph provides an overview of some of the mechanisms used by selected firms operating in the software industry. The companies chosen are highly representative because they are among the biggest corporations operating in the computer programs field and because they have been among the first to implement the *hybrid* model into their business. This analysis sheds light on the 'core' legal risks that might arise from using the *hybrid* model. It should be noted, however,

36 See note 19 above.
37 See note 29 above.
38 M Karels, "Commercializing Open Source" (July/August 2003), *QUEUE*, at 4755.

that there are several ways to develop *hybrid* models and the following samples represent but a small portion of them.

Three different groups of companies are identified:

1) Traditionally commercial (i.e. proprietary, "closed") companies that have modified their fully proprietary protection model in order to incorporate open source software. Within this group, the strategies implemented by International Business Machines (IBM) are analysed;

2) Traditionally FOSS firms that have started to include closed-types of software. Among this group the Red Hat's model is investigated;

3) Purely *hybrid* companies, i.e. software firms that have started as a mix of closed/open features. MySQL AB (nowadays owned by Oracle Corp.) is taken as a representative example for this group.

3.1.1. Closed Companies Going Open

Proprietary software companies have found that they could generate more profit and better satisfy their customers by including aspects from open source. Accordingly, they have embraced an approach based on honoring FOSS while, at the same time, relying on fees for the use of intellectual property.[39]

3.1.1. IBM: The Pioneer

IBM was one of the first proprietary companies to change its protection model and move to a *hybrid*, closed-open system.

In the 1980s, IBM was one of the most vigorous advocates of strong IP protection for computer programs. They believed that without strong IP rights, there would not have been sufficient incentives for firms to invest in software development. At that time, IBM was distributing programs merely in machine-executable form (i.e. object code form) and was using several proprietary protection mechanisms, such as copyrights, patents, trade secrets, trademarks, licensing agreements, and technical protection measures.[40] Certainly, at that time no one would have guessed that two decades later IBM would have embraced open source software.

However, IBM re-oriented themselves as an open source company, albeit to a limited degree. It currently contributes over one hundred million U.S. dollars a year to

39 See C Nosko, A Layne-Farrar, & D Garcis Swartz, "Open Source and Proprietary Software: The Search for a Profitable Middle-Ground" (2005). Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=673861.

40 See P Samuelson, "IBM's Pragmatic Embrace of Open Source", *Communications of the ACM* 49 (10), 2006. Available at: http://escholarship.org/uc/item/4xb4t1ps.

the development of Linux and other open source software projects.[41] With such a radical reversal, one cannot help but wonder what the possible motivations could have been. The answer, however, is not very straightforward and includes various considerations.

3.1.1.1. Possible Justifications

Probably the most obvious justification behind IBM's change of protection model relates to the relationship between IBM and its main competitor, Microsoft. Many companies give the "kill Microsoft" approach as a reason to pursue open source. Accordingly, companies are willing to invest in software development that might not generate monetary returns, as these efforts might impede Microsoft's ability to extract future profits by monopolizing markets.[42]

During the eighties IBM was the dominant firm in the software industry. When it entered the market for personal computers IBM decided not to build its own proprietary operating system, but to license it from Microsoft, who was a small firm at the time. In order to ensure a steady supply of programs for the PC platform, IBM required Microsoft to make interface information available to application developers.[43] The IBM PC was a huge success and soon became an industry standard.

This allowed other vendors to offer equivalent technologies, but all were required to interoperate with software created for the IBM PC. In other words: all the equivalent technologies were running Microsoft's operating system.[44] Taking advantage of this situation, Microsoft started to license its operating system to PC developers to encourage network economies. This enabled Microsoft to obtain monopoly power for its platform. Meanwhile, Microsoft started to develop Windows 3.x and soon launched Windows 3.0, which immediately became a phenomenal success in the marketplace. Microsoft's platform soon became a *de facto* industry standard.[45]

Since then, no operating system has been developed that could compete with Microsoft's offering(s). Linux is one of the first operating system with real potential to challenge Microsoft's position.[46] Therefore, investing in Linux not only allows IBM to be independent from Microsoft's licensing terms, but it also increases the chances that Linux will succeed in its competition with Microsoft in the operating system market.[47]

---

41 H Chesbrough, "The Era of Open Innovation" (2003), 44 *MIT Sloan Mgmt Rev.* 35.
42 See R Mann, "Commercializing Open Source Software: Do Property Rights Still Matter?", 20 *Harvard Journal of Law & Technology* 1 (Fall 2006).
43 See note 40 above.
44 *Ibid*. See also H Chesbrough, "Open Innovation. The New Imperative for Creating and Profiting from Technology", *Harvard Business School Press* (2003), at 93-112.
45 See P Capek, S Frank, S Gerdt, and D Shields, "A History of IBM's Open Source Involvement and Strategy", (2005) 2 *IBM Systems Journal* 44, at 249.
46 Indeed, other players are present, for e.g. the Mac OS, the iPhone OS, the JavaME, and the Android operating systems. See Operation Systems market share, September 2011 at: http://www.netmarketshare.com/report.aspx?qprid=8&qptimeframe=M&qpsp=152.
47 *Ibid*.

IBM fully entered the software market in the mid-1990s[48]. However, IBM's success was undermined by various factors, not only Microsoft's platform dominance in key platform's markets, but also by the fact that software started to be mass-marketed and became increasingly commoditized.[49]

Acknowledging these problems, IBM began thinking of some alternative business models in order to succeed and quickly discovered that what customers wanted were open standards, interoperability, and customization tailored to their own needs.With these priorities in mind, IBM concluded Linux was a better platform to meet customers' demands than traditional proprietary operating systems. Thus, it embraced a semi-open protection model accordingly.[50]

At the base of this shift in strategy towards open innovation are some of the most common reasons associated with the use of open source software. Among those, monetary advantages are probably the most important: OSS is less expensive than proprietary software and thereby reduces the overall cost that customers pay for IBM's software. [51] Furthermore, through an open innovation strategy, IBM can distribute the costs of designing, developing and improving software among many contributors. Consequently, there is less need for internal support services. In OSS projects the customers can become part of the development team, as they are willing to invest time, money, and energy on improving the software, fixing bugs, and making the code more robust and extendable. This type of distributed collaborative development is particularly relevant in a technological field like software, where the increased complexity of the technology involved and the need to integrate programs from various sources is essential to create more efficient systems.[52]

Linux provides also a common platform on which IBM can build and sell special applications and services. Consequently, IBM currently focuses on selling complementary hardware and software running on top of Linux (and other open source programs), as well as integrating other customized services to enterprise customers.[53]

Moreover, IBM sells additional complementary proprietary software to the open source product. [54] The complementary software is usually a proprietary-type of software that works in conjunction with the open source software.

Finally, an open innovation strategy allows IBM to study others' innovations allowing them to perceive opportunities for building new technologies on the open source base.[55]

---

48 It should be noted that software was not at the core of IBM initial business. Instead, IBM initially focused on developing software merely in order to sell hardware.
49 See note 40 above.
50 *Ibid*.
51 J West, and S Gallagher, "Challenges of Open Innovation: the Paradox of Firm Investment in Open-Source Software" (2006), 3 *R&D Management* 36, at 319-331.
52 *Ibid*.
53 See note 40 above.
54 See note 39 above.
55 See note 40 above.

3.2. OSS Firms Implementing IP Features

The "hybridization" process of software protection not only involves major changes from proprietary firms, but it also affects the purely open source companies. Recent years have seen traditionally open source firms starting to incorporate IP mechanisms into their protection models. Whether their motives are based on enhancing profit or whether (as they often claim) they are only a defensive tactic implemented to cope with the increasing threat from proprietary companies is an open question. Some answers can be found by analyzing the type of mechanisms implemented in these companies' practices. To this end, the next section considers the example of Red Hat.

3.2.1. Red Hat: The Pinocchio Approach

Red Hat is an American company and a well-known Linux distribution vendor. Red Hat began as a purely open source software company, but has gradually changed its policy to incorporate proprietary features, in this way embracing a *hybrid*-type of protection model.

The goal for Linux distributors is to solve one of the major problems of the Linux development model: overcoming the inconvenience of modularity.[56] Distributors do not integrate and sell the Linux components as a single package like proprietary operating systems. Instead, the individual components "tend to float around". This is due to the fact that Linux (as well as most FOSS projects) does not have one single developer, but rather group contributors. Therefore, Linux distributors, like Red Hat, consolidate these pieces in a convenient package and sell them to the consumers.[57]

Originally, Red Hat concentrated on generating revenue through supporting services and packaged products containing a manual and a CD to facilitate installation. By paying for just one Red Hat Linux package the clients also acquired the right to install it on as many computers as they wished. This type of policy, however, brought a problem typical to many open source companies [58] : inadequate revenues. To overcome this problem, Red Hat drastically changed its strategy and implemented a model that combines features of the open source and proprietary frameworks.

In 2003 the company split distribution into two different products: the Fedora project[59] and the Red Hat Enterprise Linux (RHEL)[60]. Fedora is a traditional open source project, and is to run experiments and for outside developers to submit code. RHEL is a Linux distribution system produced by Red Hat and targeted to the

---

56 *Ibid*.
57 See R Gabriel, & R Goldman, "Open Source: Beyond the Fairy Tales" (May 2002), *Perspectives on Business Innovation*, Ernst & Young, Issue 8.
58 MandrakeSoft that distributed Mandrake Linux, and SCO-Caldera, that distributed OpenLinux, are but two of many examples of OSS companies that failed due to revenues issues.
59 See http://fedoraproject.org/.
60 See http://www.redhat.com/rhel/.

commercial market. The code developed in Fedora can be included in RHEL at Red Hat's discretion. Even though the RHEL's source code is made available, the code computers need to run the operating system is conditional on purchasing a support subscription. Additionally, following a typically proprietary model, a support subscription needs to be purchased for each computer.[61]

Red Hat has also started to use trademarks as a protection mechanism. A major problem for companies trying to generate revenue from software licensed under the GPL is that because the source code is freely available, third parties can easily "compile" it for computers to read, and then resell it without having to bear the development costs.[62] To reduce their competitor's ability to obstruct its business interests, Red Hat makes use of trademark protection: another company could rebuild RHEL from freely available source code, but it would have "to strip out" all references to Red Hat to comply with trademark law.[63]

Finally, another intellectual property mechanism that is incorporated into Red Hat's model is patent protection.[64] Examples of the patents owned by the company within the field of computer programs include: the European patent EP1312195 "Method and apparatus for handling communication request at a server without context switching", the EP1691276 "System and method for verifying compatibility of computer equipment with a software product", and the EP1659493 "Replacing idle process when doing fast messages". In the U.S., Red Hat's patents consist of US2011066672 "Transaction Sticky Load Balance Policies", the US2011067007 "Automatic Thread Dumping", and the US2011249013 "Plug-in Architecture for Dynamic Font Rendering Enablement" amongst numerous others.

This move came as highly unexpected, particularly considering the company's well-known objections to software-related patents. The company has tried to reassure the public by stating that these patents are mainly for defensive purposes and used as a 'trade off' with proprietary firms that are constantly threatening them for allegedly infringing their patents. Accordingly, their policy is not to enforce those patents upon open source developers. This message is clear in the company's statement towards software patents:

> "Red Hat has consistently taken the position that software patents generally impede innovation in software development and that software patents are inconsistent with open source/free software. [...] At the same time, we are forced to live in the world as it is, and that world currently permits software patents. A relatively small number of very large companies have amassed large numbers of software patents. We believe such massive software patent portfolios are ripe for misuse because of the

61 See note 57 above.
62 J Lerner and J Tirole, "Some Simple Economics of Open Source", 50 *J. Industrial Economics* 2 (2000), 197-234.
63 See note 39 above.
64 See also Red Hat press release (June 3, 2005), "Red Hat Calls for Intellectual Property and Patent Policy reform; Red Hat Commits Significant Resources Towards Fedora Foundation, Global Reform of Government Public Policy and Advocates as Patent Commons", at: http://investors.redhat.com/releasedetail.cfm?ReleaseID=355725.

> questionable nature of many software patents generally and because of the high cost of patent litigation. One defence against such misuse is to develop a corresponding portfolio of software patents for defensive purposes. [...] At the same time, Red Hat will continue to maintain its position as an open source leader and dedicated participant in open source collaboration by extending the promise set forth below. [...] Subject to any qualifications or limitations stated herein, to the extent any party exercises a Patent Right with respect to Open Source/Free Software which reads on any claim of any patent held by Red Hat, Red Hat agrees to refrain from enforcing the infringed patent against such party for such exercise ("Our Promise").[...]".[65]

This is but a promise without any legal force and it does not exclude *a priori* the possibility for Red Hat to change its strategy at any time. Perhaps (temporary) one can find security in the general assumption that if companies enforce patents for competitive reasons (as they often do), it is not in Red Hat's interest to enforce its patents. The company would attract too much negative attention compared to the potential benefits of such action. As already mentioned, however, conditions can change and so can Red Hat's policy.

3.3. Pure *Hybrid* Firms

3.3.1. MySQL AB: The Dual Licensing Approach

MySQL is one of the world's most popular open source database systems. Originally owned by the Swedish company MySQL AB (now owned by Oracle Corp.), MySQL's *hybrid* nature lies in its original licensing model. The company used a system of "dual licensing" by combining proprietary and open source licensing models.

Generally, the dual licensing model mixes together proprietary and OSS mechanisms, and offers the same software product under both a traditional proprietary license and an open source license.[66] Technically, only one core product exists, but two licenses are used: one for free distribution and use, and another for proprietary distribution.[67]

With dual licensing anyone can download the source code for free and redistribute it, just so long as the redistributed product is licensed under an OSS license. The second license removes the open source license's restrictions and allows purchasers to distribute it and integrate it with proprietary products. This second option obviously targets companies planning to customize the product for commercial purposes. Thus, both licenses allow developers to customize MySQL and redistribute it as part of

---

65 See Red Hat, Inc., "Statement of Position and Our Promise on Software Patents" at http://www.redhat.com/legal/patent_policy.html.
66 S Comino, and F Manenti, "Dual Licensing in Open Source Software Markets" (January 1, 2010). Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=985529.
67 M Välimäki, "Dual Licensing in Open Source Software Industry" (2003), 8 *Systemes d'Information et Management* 1, pp. 63-75.

a larger product. If the larger product is released as open source, no license needs to be purchased. If the product is distributed in proprietary form, it requires a commercial license.[68]

The dual licensing system (and models where OSS is fully embedded and distributed together with proprietary software in general) includes two major and interconnected legal risks. One danger is that the OSS license (in the case of MySQL, the GNU General Public License) can dilute ownership and even eliminate the possibility to dual license, and the fact that OSS projects with multiple authors can have conflicting copyright claims poses the other.

The ability to license the product with terms other than open source, however, requires full ownership of rights to the product. Thus, no hidden liabilities in the form of code contributions from unknown third parties should remain.[69] To clear the rights and avoid legal risks, MySQL adopted a strategy that allows the firm to fully own the copyright over both the product and over all its modifications: MySQL not only develop almost all code in-house, but it also rewrites all the bug fixes and further extensions and modifications offered by external developers.[70]

Another way MySQL profited from the dual licensing system was by taking advantage of the "network effects" typical of database systems. In most cases, databases need customization to meet the specific needs of buyers. The more people use a particular package, the more developers become trained in customizing the system, inevitably leading to more written documentation and the creation of more software add-ons. This makes the system more valuable to each user, since trained developers and convenient add-on packages are easier to find and cheaper to use.[71]

In the specific case of MySQL, the GPL licensing option encouraged network effects by creating a group of developers familiar with the product. At the same time this attracted the interest of commercial users willing to pay, at least in part, because of the network effects created by the free licensing for non-commercial use under the GPL. Even though some revenue is certainly lost, (because users who download the GPL version would have, if required, purchased a license) the lost unit sales can be recouped through the higher prices charged to commercial users. In this way, MySQL utilizes price discrimination to effectively separate users according to product usage.[72]

---

68 See note 39 above.
69 See note 62 above. The issue on ownership of rights is also explained in more details later in this paper, under section 4.1.3..
70 *Ibid*.
71 See note 39 above.
72 *Ibid*.

In a recent press release, Oracle Corp. announced the addition of certain close-type features to the MySQL code.[73] This is a clear sign that things are changing and the MySQL code is not any longer fully open source software.

4. THE *HYBRID* MODEL FACES ITS LIMITS. SOME EMPIRICAL EVIDENCE

The analysis on the companies' strategies thus far shows that the *hybrid* protection models can lead to several legal risks. Companies utilize open source software either internally (for internal use) or in commercial products or services. Indeed, the legal challenges increase when a company fully embeds and releases open source code together with proprietary software. Solely internal use of open source software, such as use as part of an internal tool, is usually safe from a legal perspective. Internal use means that there is no distribution and that no FOSS code is incorporated into the company's final product.

Internal use can still be compromised by something as simple as an external contractor having access to the code. Legal risks become real once the FOSS program is distributed to a customer, contractor, or otherwise made available. They become concrete once the open source software incorporates itself as a component or as a library of the proprietary software developed by the company. This situation is particularly interesting because it raises a series of challenging legal questions. Specifically, this article analyses both the legal risks related to the FOSS licenses (i.e., compliance and compatibility with the FOSS licensing terms, risks associated with the 'copyleft' clause, and ownership of rights related risks), and the risks related to intellectual property rights (i.e., patent infringement and litigation risks, dilution of the company's own patent rights, and lack of warranties and indemnifications on the FOSS side). The focus is on the risk(s) of the simultaneous use of FOSS and proprietary software by proprietary companies, and not on the risks posed by IPRs (especially patents) to the FOSS development. This last issue has been extensively discussed in literature, and therefore, it is not the intention of this paper to further reiterate it.

Even though this article addresses the legal challenges involved, it is worth mentioning that other risks can arise from using *hybrid* protection models. For instance, the technical risks related to the OSS code (like those related to the function of the software), the security risks, the availability of support, and the compatibility of files and formats. Another highly concrete risk is the so-called "open source community risk": if a company is perceived to "violate" (in a broader sense than legal infringement) certain OSS license conditions, or even the "spirit" of that license, or goes against the generally accepted conventions of the OSS community in another way, the community may react against the company itself. The community can include not only active participants of the open source movement, but also employees of the company, employees of the company's clients, or the suppliers of the company. Overall, the community risk can attract lot of negative publicity to the company.

73 See Oracle's MySQL Blog, "New commercial extensions for MySQL Enterprise edition", at: http://blogs.oracle.com/MySQL/entry/new_commercial_extensions_for_mysql.

An empirical study under the form of a case study research composes this section. A case study analysis was chosen because an in-depth investigation was needed to provide a holistic understanding of the phenomenon. To this end, the case study relied upon a qualitative-type of analysis. A quantitative technique would have potentially obscured some of the important information that needed to be uncovered, such as whether the *hybrid* protection method is efficient and what kind of specific problems it poses in practice. The theoretical focus (i.e., the object) of the study was the problem(s) encountered by commercial companies implementing *hybrid* models of protection for their software products. The subject of the study was exemplified by representative companies who operate in the software field. A multiple case study was conducted, the reason being that more than one case was available for replication: several companies use the *hybrid* model and may encounter the problems investigated in this study.[74]

The nature of the project and the type of research questions investigated justified an 'intrinsic' and 'collective' case study. In other words, a study where the researcher has a personal interest in the case, and where a group of cases or objects are studied. The case study intended to generate new understandings, rather than answer one (of a few) specific question.[75] The questions posed to the companies included:

- *General questions,* such as 'how' do companies combine FOSS code with proprietary software? 'What' are major advantages and disadvantages for incorporating *hybrid* models (FOSS in particular) with a company's proprietary model? 'What' are the reasons for choosing certain FOSS packages? 'What' are the reasons for a company's customers for specifically asking not to incorporate FOSS pieces of code into the company's proprietary software?

- *Specific questions* on the potential *legal risks* of using *hybrid* models, such as 'how' difficult is it to review and interpret the OSS licenses terms? 'How' do these difficulties challenge compliance with all licenses used by a company? 'How' concrete is the risk of "contaminating" a company's proprietary code that becomes associated with the 'copyleft' clause, and for 'what' reasons? 'How' concrete are legal risks from the fragmentation of rights (for example, copyright) in FOSS projects? Is FOSS' rights fragmentation an advantage or disadvantage for a proprietary company implementing *hybrid* models? 'How' do companies perceive the risk that certain FOSS packages might expand to include proprietary applications? 'How' much, and in 'what' way does the incorporation of open source code into a company's proprietary software affect the risk of IP (in particular patent) infringement and litigation? 'How' concrete is the risk for the dilution of a company's patent rights, and

74 For more information on case study research see: R Yin, *Case Study Research: Design and Methods*, Volume 5 of Applied social research methods series, Sage Publications, Inc., Forth edition (2008); R Yin, *Applications of Case Study Research*, Sage Publications, Inc., Third edition (2011). See also R Stake, *The Art of Case Research*, Sage Publications, Inc., First edition (1995). See also W Tellis, "Introduction to Case Study", 3 *The Qualitative Report* 2 (1997) at: http://www.nova.edu/ssss/QR/QR3-2/tellis1.html; W Tellis, "Application of a Case Study Methodology", 3 *The Qualitative Report* 3 (1997), at: http://www.nova.edu/ssss/QR/QR3-3/tellis2.html.
75 See, for instance, Stake (1995) above.

what factors require consideration when addressing the problem? 'How' relevant is it that several FOSS products do not provide warranties or indemnifications when a company decides to choose between a proprietary and FOSS version of a component, and what are the justifications behind each option?

- *Specific questions* on possible *coping mechanisms* to navigate the legal challenges associated with *hybrid* models, such as 'what' kind of policy procedure and processes should a company implement to review the licenses it plans to incorporate? To 'what' extent does the risk of contamination of a company's proprietary code associated with the 'copyleft' clause affect the decision of a company to incorporate GPLed code (or similar types of licensed code) or not? 'What' kind of coping mechanisms can (if any) a commercial company using *hybrid* models implement to reduce the risks associated with the fragmentation of rights in FOSS? 'How' can a company effectively monitor and prevent the possible infringement and litigation risks associated with the *hybrid* models in general, and with the incorporation of open software code into the company's final proprietary software in particular? 'How' can a company effectively monitor and prevent possible dilutions of its own patent rights when it releases software under FOSS licenses?[76]

To select the subjects of investigation, an information-oriented technique rather than random sampling was used. [77] The companies were chosen for their representativeness with respect to the overall purpose of the study's research objective (i.e., they were 'key' cases) and to maximize what could be learned in the period of time available for the study. Specifically, the study considered six cases, composed of one consultancy company and five software companies. These specific cases were chosen for the following reasons:

- The field of operation of the companies: they were all companies whose main business lay in software, i.e. companies whose innovation was based on the software they developed as opposed to companies that used software indirectly. The consultancy company operated specifically in the field of FOSS for proprietary companies, and therefore, considered a valuable source of insightful information from a more neutral perspective.

- All the companies implemented or dealt with *hybrid* models of protection.

- Key people working at these companies were well-informed experts on the research object and in possession of important knowledge.

The size of the companies and their geographical areas of operation were not regarded important factors in the selection of the cases.

76 For more information see Appendix V: "Interview Questions, Software Companies Implementing *Hybrid* Protection Models".
77 See note 74 above.

The study used two different sources of evidence: documents and interviews. Other sources of evidence usually considered in case study research such as, archival records, direct observation, participant observation, and physical artifacts were not relevant for this study.[78]

The documents used were mostly academic literature (legal and economics), case law, legislations, publicly available companies' policies in the fields of intellectual property and open source, companies' websites, and newspaper articles.

Interviews were the most important source of information for the study, and followed an open-ended format.[79] All the interviews were conducted during the autumn of 2011. Key respondents were asked to comment on the research questions from the perspective of their own company, but also and more importantly, based on their extensive knowledge of the field. The respondents were free to propose solutions or provide insights into the subject matter, as well as to corroborate evidence obtained from other sources. Indeed, this 'open' method expanded the depth of the relevant data gathered.

A draft report was written based on both the documents consulted and the answers received in the interviews.[80] All the participants in the study then reviewed the report to verify the accuracy in reporting their answers, as well as the overall conclusions and observations. Several external peers then critically challenged the results and provided relevant feedback in a discussion regarding the report. This process enhanced the accuracy of the case study.

The anonymity of the people interviewed and their respective companies was necessary due to the participants' consideration of the topic as being both controversial and confidential. As a compromise, a cross-case analysis was composed instead of a single-case report.[81] The case study report does not portray any single one of the companies interviewed, but rather a synthesis of the lessons learned from all of them. Accordingly, none of the cases are presented as single-case studies. Instead, examples from the cases are discussed under each research topic section (i.e. 'Compliance and Compatibility', 'Reciprocity of FOSS Licenses', 'Ownership of Rights', 'Patent Infringement and Litigation', 'Dilution of the Company's Own Patents', and 'Lack of Warranties and Indemnifications'). Furthermore, there are two sub-sections in each section: 'Concerns' and 'Coping Mechanisms'. Under the section 'Concerns' theoretical frameworks are discussed, while the 'Coping Mechanisms' section presents data from the empirical study. In addition, the research topic sections 'Reciprocity of FOSS Licenses' and 'Patent Infringement and Litigation' include two additional sections: 'Free and Open Source Software Litigation' and 'Patent Litigation on FOSS'. Relevant case law on FOSS and copyright and on FOSS and patents is discussed in these sections.

78 *Ibid*.
79 *Ibid*.
80 See Yin (2008), note 74 above, at 141-167.
81 *Ibid*, at 170-173.

4.1. Licensing-Related Concerns

A major set of legal problems associated with the *hybrid* protection model includes licensing-related issues. The following paragraphs focus on compliance with the OSS licensing terms and the compatibility among the licenses, the issues related to the 'copyleft' clause, and the ownership of rights-related problems.

4.1.1. Compliance and Compatibility

Concerns

One main point of concern with the *hybrid* model is the compliance and the compatibility between the OSS and the proprietary licenses' terms and conditions. Two separate sets of problems should be distinguished: on the one hand, problems might derive from the use of both proprietary (either third parties' proprietary software or the company's own proprietary licensed code) and OSS licensing models; on the other hand, incompatibilities might arise between OSS licenses themselves. Although the increase in the number of open source licenses has improved firms and project leaders ability to find models that better suit their needs, such a multiplication has also led to the creation of licenses that are incompatible with others. Ironically, this might distort the original purpose of open source software by limiting, rather than encouraging, the reuse of code.

Coping Mechanisms

The companies interviewed cited compliance with the obligations imposed by licenses to be the biggest and most concrete reason for concern. The basic difference between commercial licenses and FOSS licenses is the fact that with the FOSS licenses the terms cannot be changed, while in a commercial set up, there is usually some room for negotiation on the licenses' terms. To minimize the license-related risks all the companies agree that it is extremely important to build a solid design of the architecture and principles around the types of licenses the company plans to incorporate (e.g. where should the company include GPLed components and where it should incorporate components licensed under more permissive OSS licenses). In accordance with this consensus, all the interviewed companies affirmed to conduct some sort of compliance and compatibility checks of the licenses included in their models. The medium and large companies all had an internal procedure for assuring compliance of the licenses in use. Even though the start-ups and small companies interviewed did not have well-structured policies on the matter, they usually had at least one person in-house responsible for scrutinizing the licenses' compatibilities and compliance with the licenses terms.

According to some of the companies, a very important requirement for choosing certain OSS packages instead of others is the compatibility of the OSS licenses with each others, as well as third parties' proprietary code and the company's own proprietary software licenses. A common problem most of the companies reported in the licenses' review process was the difficulty in interpreting the OSS licenses'

terms. This was attributed to several reasons: the language of the licenses *per se*, the fact that multiple versions of the licenses can exist for the same piece of code, thereby making it difficult to detect which version actually applies, and the fact that the same OSS code is often licensed under several different OSS licenses, often stating different rules and making it challenging to understand the applicable principles.

Only some of the firms acknowledged conducting further checks on the licenses' terms every time the FOSS packages receive updates. One company was very aware of risks involved with possible changes to the licenses in the FOSS packages. While reviewing the licenses included in an FOSS package the company was using, the company discovered the package was extended with certain proprietary features. As the company was using these features, it found itself in a situation of either having to change the FOSS package or having to pay the license fees for the proprietary parts of the code. As the particular component was not an essential part of the company's product, the decision was made to remove the full package and substitute it with another FOSS component. The company reported that after this experience, they introduced a more thorough review of the licenses' terms.

4.1.2. Reciprocity of FOSS Licenses

Concerns

The license restrictions of the open source software have a clear impact on a proprietary company's strategy. One of the biggest risks is that a companies' proprietary code will be "forced" open. The risk appears particularly high with the 'copyleft' licenses in general, and with the GPL in particular.

The 'copyleft' licenses are typically quite restrictive when it comes to combining proprietary and open source types of software. According to the GPL, if a piece of code that in whole or in part contains or is derived from an OSS code or any part thereof is distributed or published together with another (proprietary) software, the source code of the entire final product must be made available and licensed under the terms of the GPL.[82] In other words, if a company combines GPLed software with its own developed proprietary software, the question comes down to whether or not the result is published "as a whole work". Technical issues will probably determine what is considered as "a whole", like how closely the programs interact, how they are linked together, and how the proprietary program loads with the GPL-licensed code. On the other hand, if the proprietary code that a company combines with the GPL code is apparent and recognizable as an independent and separate work, such code might be able to remain free of the GPL "taint".

82 See note 10 above.

Furthermore, the GPL includes also specific patent clauses[83] with the intent "to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary"[84]. In other words, incorporating code originally acquired under a GPL-type of license might dilute possibilities for commercialization and ultimately compromise the company's IP rights.

The BSD and academic licenses give more flexibility and opportunity for commercial exploitation. Under the BSD-type of licenses, licensees can generally distribute their derivative works without any obligation of source code disclosure. This means that licensees are free to integrate FOSS code (as well as modifications of the code) into proprietary software, and then redistribute the whole piece of software under proprietary licenses.

Free and Open Source Software Litigation

Problems related to the enforceability of the FOSS licenses have become a hot topic of discussion in recent years, especially with the uncertainty surrounding the application of the FOSS licenses' principles. In an already risky and difficult corporate environment, companies are attempting to minimize their potential liabilities, and one way of doing this is by reducing the companies' legal risks.[85]

Most GPL cases thus far have been on copyright issues focusing on failures to comply with the source code's distribution requirement. In general, the retention of the copyright for the original code allows the enforcement of the FOSS licenses: the original developer retains the copyright for the original program and subsequent developers retain the copyright(s) on their improvements. If one does not comply with the license it terminates and it becomes impossible to copy, modify, distribute, or redistribute the code without violating the owners' copyrights.[86]

In the US, court cases on the interpretation of the FOSS licensing terms started to appear in early 2000. For instance, *Palnetary Motion vs. Techsplosion* (2001)[87], *Progress Software Corp. vs. MySQL AB* (2002)[88], and *Computer Associates vs. Quest* (2004)[89] all centered on the enforceability of the GPL licensing terms. Even though they followed different paths, all decisions presumed that the GPL terms are binding.

---

83 For instance, see clause 7 of the GNU General Public License, version 2: http://www.gnu.org/licenses/gpl-2.0.html and clause 11 of the GNU General Public License, version 3: http://www.gnu.org/copyleft/gpl.html.

84 The preamble of the GPL version 2 explained the motivation behind the patent clause: "[…] every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free." See GNU General Public License, version 3.

85 See A Guadamuz, "Legal Challenges to open source licenses", (2005) 2:2 *SCRIPT-ed*, 301-308.

86 See GNU General Public License, Version 3 (2007), note 10 above.

87 *Palnetary Motion, Inc. vs. Techsplosion, Inc.* (2001) United States Court of Appeal for the Eleventh Circuit 261 F.3d 1188.

88 *Progress Software Corp. vs. MySQL AB* (2001), Civil Action No. 01-11031 PBS.

89 *Computer Associates, Inc., vs. Quest Software, Inc., et. al.* (2004) No. 02 C 7421.

SCO vs. Linux[90] are a series of legal disputes between the company SCO Group (a well known software developer of the UNIX related products) and several Linux vendors and users, including IBM, Red Hat, and Novell. Since 2003, SCO has claimed that these companies infringed upon SCO's intellectual property on the UNIX kernel. Despite the fact that outcomes for some of SCO's cases are pending, they certainly increased the financial importance of all 'copyleft' licenses.[91]

Many of the court proceedings refer to the Software Freedom Law Center (SFLC). Launched in 2005, it provides *pro-bono* legal representation and related-services to non-profit developers of free and open source software. Among the lawsuits filed by the SFLC are the BusyBox-related litigations: starting from 2007, the SFLC filed a series of copyright infringement suits against various defendants on behalf of BusyBox's principle developers, Erik Anderson and Rob Landley. They claimed violation of the GPLv2. Lawsuits were filed, among others, against Monsoon Multimedia Inc., Xterasys Corp., High Gain Antennas LLC, Verizon Communications Inc., and Cisco System Inc. All the cases centered on the failure of the infringer to distribute the source code under the terms of the GPL license. Most of the cases ended in settlements and with the defendants agreement to start distributing in compliance with the license and paying the corresponding fees.

A complex but interesting dispute is the *Jacobsen v. Katzer* case[92]. The dispute involved copyright and patent issues (the patent issue, however, was subsequently removed from the case), as well as Digital Millennium Copyright Act and 'cybersquatting' issues. The importance of this case lies in the CAFC strengthening artistic license agreements by affirming (for the first time in history) that such licenses' violations equates copyright violations. Even though the reasoning was limited to the artistic license and subsequent interpretations of each open source license will depend on its precise provisions, this decision has strong repercussions for FOSS licenses in general.

In Europe, Germany has developed what is perhaps the most comprehensive body of FOSS-related case law. This is partly due to one important project that was launched in 2004 by a German programmer, Harald Welte: the GPL-Violations.org[93] project. The purpose of the project was to track down and prosecute violators of the GPL. Since 2004, GPL-Violations.org claims to have enforced over one hundred actions that were successful in either settling or obtaining judgments. The most relevant cases include: the *Welte vs. Sitecome Germany* from 2004[94], the *Welte vs. D-Link*[95] from 2006, and the *Welte vs. Skype*[96] from 2008. The first two cases confirmed

90 These cases included: *SCO-Caldera vs IBM* (2003) US District Court District Court of Utah, No. 2-03-cv-294; *SCO-Caldera v DaimlerChrysler* (2004) Oakland Country Circuit Court, Michigan, No.: 2004-056587; *SCO-Caldera v AutoZone* (2004) US District Court District Court of Nevada, No.: 2-04-cv-00237-RCJ-GWF; *Red Hat vs. SCO-Caldera* (2004) US Federal Court District of Delaware, No: 1-03-cv-772; *SCO-Caldera vs. Novell* (2004) US District Court District Court of Utah, No.: 2:04cv0139. All the legal documents related to the cases can be found at: http://sco.tuxrocks.com.

91 See note 85 above.

92 *Jacobsen vs. Katze* (2008) Federal Circuit, No.: 2008-1001.

93 See http://www.gpl-violations.org/.

94 *In Re Welte vs. Sitecome Germany*, District Court of Münhen I , No 21 0 6123/04 (2004).

95 *In Re Welte vs. D-Link Germany*, District Court of Frankfurt am Main, No 2-6 0 224/06 (2006).

that failure to provide source code originally licensed under the GPL is a violation of the GPL's terms, and ultimately warrants legal action. In the Sitecome's case the court granted a preliminary injunction against Sitecome for failing to provide the source code under the terms of the GPL. Specifically, the Court rejected the defendants' claims that it had not agreed to the GPL and that the plaintiff had waived all rights to the code by distributing it under the GPL. In the D-Link case the court affirmed the validity of the GPL terms under German law and ordered D-Link to reimburse GPL-violations.org for the enforcement expenses. This was one of the first rulings on damages arising from a GPL violation. Finally, in *Welte vs. Skype*, Skype sold third-party hardware on its website unaccompanied by the source code or a copy of the license. Skype tried to remedy this by providing links to both the source code and the license, but this was insufficient. According to the Judge: "If a publisher wants to publish a book of an author that wants his book only to be published in a green envelope, then that might seem odd to you, but still you will have to do it as long as you want to publish the book and have no other agreement in place"[97]. In other words: full compliance with the GPL is needed.

As shown, all the existing case law has focused on the distributor's failure to make available certain FOSS code, and not on the allegedly "tainted" or "contaminated" company's proprietary code.[98] When companies have been found to distribute in breach of the GPL licenses' terms, they have been asked to either start distributing in compliance with the GPL (and pay some license fees) or to stop distributing but not to distribute their proprietary code under the GPL.

It should be pointed out that it appears to be a widely accepted procedure among FOSS right holders to initially attempt to find a reasonable deal with the alleged infringer. Usually the right holder would ask the company in violation to comply with the license before initiating any legal proceeding. For instance, In *FSF vs. Cisco*[99] FSF initiated legal proceedings after several attempts to elicit compliance with the license terms, and several refusals by Cisco in taking any action.

Coping mechanisms

The legal risks associated with the *reciprocity of the FOSS licenses* in general, and the *copyleft clause* in particular, have been reported as highly challenging and one of the most concrete risks associated with the *hybrid* model. To reduce such risks, companies have developed different coping mechanisms. Many of the interviewed companies shared a common practice: to avoid using OSS packages which includes GPL or copyleft-type of licenses. Some companies included GPLed (or GPLed-type) code, but only for OSS tools they used internally. These companies ensure that pieces of GPLed code are not distributed to the customers. One of the

96 *Welte vs. Skype Technologies SA*, Higher Regional Court of Munich (2008).
97 This English translation was provided in Harald Welte's blog:
http://laforge.gnumonks.org/weblog/2008/ (08 May 2008).
98 One exception is the *Progress Software Corp. vs . MySQL AB* (2001), Civil Action No. 01-11031 PBS mentioned before in note 88.
99 For instance in *FSF vs. Cisco* (USA 2008) before initiating legal proceeding, the OSS right holder had asked Cisco for over two years to start distributing in compliance. The case was settled later in 2009.

companies also claimed to receive requests from its own customers to omit FOSS code from their products, due to risks associated with the 'copyleft' clause.

One company avoids incorporating GPLed software into its final product because of the so called "community risk". Specifically, the company believes that the OSS community has tools to detect possible violations of the OSS licenses' terms. If the OSS community considers the company non-compliant with the GPL, the company could attract a lot of negative attention.

Only one company distributes GPLed code with its final proprietary software. According to the company, interpreting something as unclear as the GPL licenses (and FOSS licenses in general) in a purely literal, legal interpretation is not accurate. Instead, it is essential to actively follow interpretations given by different sources, including the existing and evolving jurisprudence, the more recent interpretations of organizations like the Free Software Foundation (FSF), and the FOSS community. The company was well aware of the fact that none of the court decisions have focused on the possibility that GPL terms might "taint" a company's proprietary code, and that the primary aim of any FOSS right holder is to reach an agreement outside the courtroom. The company justifies this with the fact that the primary aim of the FSF and the FOSS community is to promote the philosophy and ideas behind the GPL and open source software. This goal has been achieved by seeking compliance of the source code distribution both in and out of the courtrooms. For example, there have been convincing court decisions confirming the ideas and philosophy of the open source movement, and this promotes the use of open source software in general. The company maintains that this success is attributable to the FSF and the FOSS communities focus on implications of the license, such as the "contamination" of the proprietary code issue. The company is not considering changing this strategy in the future, due to the long lasting acceptance of this interpretation of the 'copyleft' clause. Consequently, it does not possess any back up plan.

4.1.3. Ownership of Rights

Concerns

As mentioned earlier, another concern associated with *hybrid* models relates to the ownership of rights. To commercialise software, companies must have undisputed rights over the product. Open source licenses are copyright licenses, and not interpreted as the licensor relinquishing rights. In order to incorporate an OSS licensed piece of software into a proprietary framework, a company must carefully evaluate the conditions under which such a product had been licensed and acquire all the rights over it.

Ownership of rights issues might concern the proprietary companies that use OSS due to the collaborative way open source projects are developed and the resulting fragmentation of copyright rights. For instance, it might often be difficult to identify the right owners in a FOSS project, as there might be several holders for a single FOSS

component. Furthermore, FOSS components might include files without copyright notices, or notices without proper licenses.

Coping mechanisms

Theoretically, the *ownership of rights* concerns can be solved in two ways: 1) by fully re-writing the FOSS code (including re-write all the contributions to the code, for instance for bug fixes), or 2) by somehow acquiring all the necessary rights over the software.

The first option, adopted (for example) by MySQL, is safe from a legal point of view. Fully re-writing the code, however, might be prohibitively expensive as it may involve complex and costly R&D studies,[100] especially since software is constantly becoming increasingly more complex. As mentioned earlier, digging into big FOSS packages to change parts of the code can be very laborious and time consuming, particularly if a programmer is not very familiar with such software.

Another option to solve the issue of rights ownership is to try to obtain all necessary rights through a specific license or contract. Even though this offers a more affordable option, such an alternative might not clear all the legal risks. Problems can remain if the transfer of rights is incomplete if, for instance, because the code contributor has no authorization to withhold the necessary rights.[101]

Most of the interviewed firms had not thought the issue through. Some of them agreed that the ownership of rights poses a risk, but they did not consider it highly problematic in practice. None of the companies are taking any precautions in this respect. One of the companies affirmed that proper compliance with the FOSS licenses included in the packages suffice in clearing all the risks of ownership.

One company considered the fact that in many FOSS projects the fragmentation of rights is a benefit for the FOSS users (i.e. the companies). The fragmentation of rights limits the possibility for the copyright holders to act because it is too challenging for a single FOSS right holder to get consensuses to pursue certain claims. The company recognizes that this justification does not 'guard' against all the risks because problems with the ownership of rights are highly contingent upon jurisdiction. For example, in certain jurisdictions it is possible to pursue a claim even in the case of "partial" ownership.[102]

4.2. IPRs-Related Risks

Another set of interesting legal challenges relates to the intellectual property related risks. Specifically, the patent infringement and litigation risks, the dilution of the

100 See M Välimäki, "Dual Licensing in Open Source Software Industry", (2003) *Systemes d' Information et Management* Vol. 8, No. 1, pp. 63-75.
101 *Ibid*.
102 See, for instance, the case of Finland: Finnish Copyright Act (2010), Section 6. Available in English at: http://www.finlex.fi/en/laki/kaannokset/1961/en19610404.pdf.

company's own patents and the lack of warranties and indemnifications from the FOSS side.

4.2.1. Patent Infringement and Litigation

Concerns

Intellectual property law does not discriminate between proprietary and open source software for enforcement purposes. Open source is not inherently more likely to infringe upon software patents than proprietary software, and vice versa. Software faces potential IP infringements because it is highly complex, the patent rights in the field are innumerous, many of the patents available are either very broad, or not novel, or not inventive; therefore even a single small software package can infringe upon several hundred distinctive intellectual property holdings.[103] These problems are common to all types of software.

The actual risk of litigation (and the directly related risk of getting injunctive relives), impacts the open source firms in a different way than the purely proprietary and *hybrid* companies.

It has been argued that risks of violations are higher on the open source side: open source developers often operate outside the IP legal framework that dominates the proprietary software industry because most open source projects lack the infrastructure to properly monitor their code base. Furthermore, these organisations often accept code contributions from developers who are unknown to the open source community and therefore have little control over the origin of the code.[104]

Additionally, the fact that the source code is open in FOSS enhances the possibility for competitors to detect infringement. In software, third parties are not always able to perceive patent infringements from the outer function of the product; sometimes it might not be possible to detect violations without knowing exactly how the product works. Generally, the closer the patent is to pure software, the more difficult it is to see from the outside how the software works. In these cases, open codes can prove essential to detecting patent infringements.

On the other hand, the risk of litigation appears higher for both purely proprietary and *hybrid* software companies than for the purely open source companies. IP infringement by an open source software package is more likely to incur legal action from a commercial company, than vice-versa. Even though open source software development has been halted in specific areas of technology with known software patents (e.g. MP3 audio, LZW data compression present in the GIF graphics file formats, etc…) no FOSS organization has yet been subject to legal action for patent infringement. One important reason for this is that pure open source software companies are generally 'not worth suing'. Another is that widely used FOSS projects

103 For more info see notes 5 and 7 above.
104 On this concern, however, it is worth noting that the same problem is found also among proprietary companies. See, for e.g., M Lemley, "Ignoring Patents", 2008 *Mitch. St. L. Rev.* 19, 19-34.

with a large community are more aware about the potential to infringe on existing claims. In other words, the fact that many people use the same FOSS software lowers the chances for companies to be hit by a claim.

The issue of patent infringement is particularly complex for companies implementing *hybrid* models. One key reason that might increase the litigation risks for *hybrid*-type of companies relates to the fact that, as already mentioned, OSS packages often have several users. This means that if company X succeeds in proving that company Y is infringing on patents for a certain piece of OSS code, company X can confidently assume that it can conduct successful legal proceedings against other companies implementing the same OSS software package as company Y. This factor might increase the litigation risks for companies implementing *hybrid* models. The *SCO Group vs. Linux* controversies[105] launched in 2004 and all the cases starting from 2010 in the United States over the Java and the Android operating systems (that will be exposed later in the paper) represent emblematic examples of such risks.

The more competitive a company is and the bigger the market share it owns, the higher the risk of being hit by a patent claim. The reason why certain types of software (e.g. software in phones) are highly litigated is most certainly because they are associated with highly successful devices. Furthermore, if the devices (in particular OSS based devices) are delivered directly to consumers the litigation risks increase because the products become more visible. In other words, successful consumer products have an increased risk of litigation for companies using the *hybrid* protection model.

Finally, it is worth mentioning that in the case of *hybrid* models (in the same way as for proprietary software in general), litigation risks depend on the jurisdiction. For example, it is well know that litigation risks are particularly high in the United States, due to its long history of allowing the patenting of software. In Europe, the case law has limits under EPC Art. 52's exclusions of computer programs "as such" from patent law.[106]

Patent Litigation on FOSS

On the specific issue of FOSS and patents, the existing case law is relatively limited and mostly comes from the USA.

Barracuda vs. Trend Micro (2007)[107] relates to the infringement suit filed by the antivirus software vendor Trend Micro against Barracuda Networks for Barracuda's use of the open source ClamAV product in its network gateway protection devices. The claim is that ClamAV violates one of Barracuda's patents, filed in 1995[108]. Trend Micro accused Barracuda of infringing its patent directly, contributory, and by inducement. Barracuda went to a Californian federal court first and filed a lawsuit

105 *Caldera Sys., Inc. vs. Int'l Bus. Machs. Corp.* (2003) US District Court District Court of Utah , No. 03-CV-0294.
106 For more information see RM Ballardini, note 7 above.
107 *Barracuda, Inc., vs. Trend Micro, Inc.* (2007) USITC, No. 337-TA-624, 72 Fed. Reg. 74, 329.
108 US patent 5,623,600 - "Virus detection and removal apparatus for computer networks".

against Trend Micro[109], seeking to settle the controversy through a declaratory judgment declaring Trend's patents invalid. With the help of the FOSS community, Barracuda is now trying to gather prior art in order to invalidate Trend's patent.

In 2006 FireStar sued Red Hat for patent infringement in *FireStar vs. Red Hat*[110]. The dispute was settled in June 2008[111] with the agreement that Red Hat claims afford broad upstream and downstream protection for the whole FOSS community. *Microsoft vs. Tom Tom*[112], saw the software giant accusing Tom Tom's navigation products of infringing upon Microsoft's patents for the FAT32 file system, was settled in March 2009.

The series of legal disputes related to the Java and Android platforms require discussion. These lawsuits began in 2010 with *Apple vs. HTC*[113] (Apple claimed that HTC infringed upon twenty patents in the iPhone's user interface), with Apple claiming different patent infringements by the Android open source operating system. Other Android related lawsuits are *Microsoft vs. Motorola*[114], *Apple vs. Samsung*[115], *Microsoft vs. Barnes and Noble*[116], and *Oracle vs. Google*[117]. Specifically, in this last case filed in August 2010 at the District Court for the Northern District of California, Oracle claimed willful infringements of certain patents related to the Java programming language distributed on Google's developed Android software, and of some unspecified copyright rights. At the time of writing the cases are still pending.

Coping mechanisms

Theory

The academic literature suggests that to mitigate the risk of patent litigation, several coping mechanisms are available to companies. For instance: patent acquisition, re-engineering sections that allegedly infringe patents (when possible), collecting and keeping prior art information to invalidate patents either in-house or through big

109 *Barracuda, Inc., vs. Trend Micro, Inc.* (2007), US District Court Northern District of California (San Jose) No.: 3:07-cv-01806-MHP.
110 *FireStar Software, Inc., vs. Red Hat, Inc., et al.* (2006) US District Court Texas Eastern District Court, No. 2-06cv-258.
111 See Settlement Agreement, 6 June 2008, available at: http://www.redhat.com/f/pdf/blog/patent_settlement_agreement.pdf.
112 *Microsoft Corp. vs. Tom Tom NV and Tom Tom, Inc.*, (2009), US District Court Western District of Washington at Seattle.
113 *Apple Inc., vs. High Tech Computer Corp., a/k/a HTC Corp., HTC (B.V.I.) Corp., HTC America, Inc., Exedea, Inc.* (2010) US District Court District of Delaware.
114 *Microsoft Corp. vs. Motorola, Inc.* (2010), US District Court Western District of Washington at Seattle.
115 *Apple Inc., vs. SAMSUNG ELECTONICS CO. LTD, et al.* (2011) US District Court California Northern District (Oakland) No.: cv-11846.
116 *Microsoft Corp. vs. Barnes and Noble, Inc., et al.* (2011) US District Court Western District of Washington at Seattle.
117 *Oracle America, Inc., vs. Google Inc.* (2010), US District Court California Northern District (Oakland), No.: 3:10-cv-3561.

projects (like the Open Source as prior art project[118]), actively participating in projects that facilitate better examination and quality of patents (e.g. the "peer-to-patent" project[119], the "patent quality index" project[120]), or establishing patent pools with reasonable terms.

On a broader scale, there are several recent initiatives to turn patents into "open source intellectual property rights", granting use to all members of the community.[121] Commitments to encourage the development of OSS projects, such as patent holders unilaterally pledging not to enforce some of their patents against users of certain open source software, are one example of such initiatives. A typical example of this model was the agreement reached in 2006 between Microsoft and Novell, where Microsoft announced not to enforce its patents against the version of Linux distributed by Novell.[122] Another initiative relates to patent owners committing not to sue those adhering to a statement of permitted use. A plan put forward by IBM in 2005 was formulated along these lines, where the firm announced that it would release 500 of its patents into a "patent commons" available for the open source community. [123] Other initiatives aim at coordinating and encouraging unilateral commitments. For instance, following the aforementioned decision by IBM, several companies have made similar patent pledges. These companies include the Open Source Development Lab (OSDL), a non-profit institution financed by large commercial companies, and dedicated to the promotion of Linux among firms that now host their patents in a "patent commons".[124] In other words, the OSDL provides a central location for patent pledges and software patents. Nokia, for one, has committed not to assert all its patents against the Linux kernel.[125] In 2007 the OSDL merged with the Free Standards Group (anon-profit consortium chartered to specify and drive the adoption of open source standards) to form The Linux Foundation. Both organisations narrowed their focus to promoting Linux in competition with Microsoft Windows.[126] Theoretically, patent commons do not only benefit the OSS developers by providing them with a shelter, but they also reduce the litigation costs: companies participating in the commons cannot benefit from the protection offered by the commons unless they agree not to sue other firms or beneficiaries for infringement.

118 See http://www.linuxfoundation.org/programs/legal/osapa.
119 See http://peertopatent.org/.
120 See http://www.law.upenn.edu/blogs/polk/pqi/faq.html.
121 See, for instance, note 118, 119 and 120 above.
122 See Microsoft News Center, "Microsoft and Novell Announce Broad Collaboration on Windows and Linux Interoperability and Support" (Nov. 2, 2006), at:
http://www.microsoft.com/presspass/press/2006/nov06/11-02MSNovellPR.mspx.
123 See "IBM Statement of Non-Assertion of Named Patents against OSS", at:
http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf.
124 See www.patentcommons.org.
125 See Nokia Corp. press releases, "Nokia Announces Patent Support to the Linux Kernel", (May 25, 2005), at: http://press.nokia.com/2005/05/25/nokia-announces-patent-support-to-the-linux-kernel/.
126 See http://www.linuxfoundation.org/.

It is worth mentioning that in recent years web initiatives such as Groklaw[127] and the Foundation for a Free Information Infrastructure (FFII)[128], have been launched. Their goal is to increase communal awareness on the issues concerning IPRs and OSS.

Practice

None of the interviewed companies considered that the use of open source software in their final products enhance (or reduce) the risk of patent litigation.

According to some of the companies, one essential characteristic of the OSS packages they use is to have a large OSS community as support. Highly used OSS packages assure companies that the community of users would promptly detect possible patent claims. Furthermore, these companies were confident that in the case of a potential patent infringement, a large OSS community would find ways of dealing with the issue, such as by re-writing the code in question. Most companies thought that the risk of being hit by IP claims (in relation to the OSS code they incorporate) is higher if using OSS packages that are, in one respect or another, unpopular.

One company considered the fact that the open FOSS code does not necessarily lead to higher litigation risks. According to this company, the fact that in proprietary software the actual code is 'closed' (i.e. not publicly disclosed) can trigger more litigation. The infringements based on analysis rather than facts (based on the source code) leaves more room for interpretation.

Most of the companies were not concerned about facing patent claims on the FOSS code they use. Some companies relied on the fact that their biggest and most successful competitors will make better targets in any patent claims regarding their use of FOSS code. These companies affirmed that in such a situation, they would immediately change the problematic FOSS component. This operation can be highly expensive and time consuming or pretty simple, depending on the importance of the component. Only one of the interviewed companies, however, admitted to regularly monitoring the FOSS packages they are using for potential IP claims. Another company only follows the biggest infringement cases related to FOSS and patents.

One company considered that the incorporation of FOSS software into a company's model can play a positive role in an alledged infringement case. In legal disputes, the open source community can gather prior art to defeat claims on the defendant's behalf. The active participation of the community depends on the specific case and on the position and reputation of the alleged infringer.

Most of the interviewed companies shared the opinion that the patent litigation risks for FOSS within the framework of the *hybrid* protection model are relatively low: taking into account that almost every company in the world that produces software uses certain portions of FOSS, the amount of IP litigation is very minimal. They generally agreed that the litigation risk is higher when a company delivers its

127 See http://www.groklaw.net/.
128 See http://ffii.org/.

products directly to the consumers and is very successful. Some of the companies, however, were not convinced that the litigation risk is different for companies that implement *hybrid* models than for those that use only proprietary software or FOSS as a strictly internal tool.

One company considered the area of industry where the company operates as a key feature when it comes to patent litigation. The company specifies the telecommunications sector as the riskiest area of technology for patent litigation.

Another company identified a company's jurisdiction as an important factor for patent litigation risks. The United States was considered much more dangerous than Europe in this regard. For instance, this specific company does not currently own any patent on the software it produces, but would not consider operating in the USA without filing some patent application..

Finally, one company opined that patent litigation is an incentive in commercial disputes; disputes that have their own logic and do not relate to open source software. The size of companies' patent portfolios and the commercial situation among competitors usually plays a part in triggering patent litigation. FOSS does not play any active or important role in this scenario.

Overall, none of the interviewed companies considered the risk of patent litigation in particular relation to the OSS part of the code they use, as very concrete and, accordingly, do not implement any specific coping mechanism on this respect.

4.2.2. Dilution of the Company's Own Patents

Concerns

Another problem with *hybrid* models is the dilution of the company's own patent rights. Dilution may result from an explicit or implied patent license under the applicable FOSS license. Determining the patent portfolio's exposure to the FOSS licensing model may be difficult. For instance, both the GPLv3 and the BSD license (i.e. the most used FOSS licenses) fail to mention patents. Notwithstanding the absence of an explicit patent grant, however, FOSS licenses may include implied patent grants. These provisions state the right holder (or the distributor), under his/her authorization, may implicitly grant a license to the recipients of the components to practice any right holder's patent claims covered by such component.[129]

Implied patent licenses can be deduced from any conduct of the right holder that induces reasonable belief in the existence of such a license. These can include any written statement of the patent holder (e.g. the wording of the open source license) and/or the way the patent holder acts. For instance, the fact that the BSD license grants the right to "use, modify, copy, create derivative works and distribute the software" might induce reliance based on the statements of the right holder.

129 See A Pugh, and L Majerus, "Potential Defences of Implied Patent License Under the GPL", *Fenwick and West LLP* (2006).

Furthermore, releasing the code under any FOSS license (which imply that the software is "free" for everybody to be used, copied, distributed and redistributed), might, in effect, induce reasonable reliance on an implied patent grant based on the acts of the patent holder.[130]

Indeed, the interpretation of an implied patent license is regional, and not all jurisdictions recognize such a concept as part of their legal regimes. Thus, the place where the software is released and/or where the patent is granted should also be taken into account when evaluating the risk of dilution.

Coping mechanisms

Only one of the interviewed companies released products under FOSS licenses. The company did not consider the risk of dilution as highly problematic. The company was confident that an accurate and careful internal review on both the explicit and the implicit patent grants included in the FOSS licenses used, as well as consideration of the national interpretation of the laws of the country of operation of the company, successfully minimize risks associated with dilution of patent rights.

4.2.3. Lack of Warranties and Indemnifications

Concerns

The fact that most FOSS providers do not offer the same warranty protections typically given to commercial products might represent an additional source of concern when companies consider implementing the *hybrid* protection model. Some OSS organizations have proposed such warranties. Hewlett-Packard, has announced that it will offer legal protection (albeit with rather strict conditions) for its version of Linux. [131] Similar programs have been implemented by other OSS companies, such as Red Hat[132], Novell[133], Hewlett-Packard[134], and JBoss[135].

Coping mechanisms

Most of the interviewed companies feel this risk is not something specific to open source software. Generally, companies did not feel safer by using proprietary

130 A Haapanen, "The is No Such Thing as Free Lunch – Implied Patent Grant Under Open Source Software Copyright Licenses", *Law in The Internet Society* (2008), at: http://moglen.law.columbia.edu/twiki/bin/view/LawNetSoc/AnnaHaapanenPaper1.
131 See Hewlett-Packard Company Website – Terms of Use and Legal Restrictions, at: http://www8.hp.com/us/en/privacy/terms-of-use.html.
132 See Intellectual Property warranty, Red Hat, Inc., at: http://www.redhat.com/legal/open_source_assurance_agreement.html.
133 See Linux indemnification program Novell, Inc., at: http://www.novell.com/licensing/ntap/.
134 See Linux indemnification for HP customers at: http://h71028.www7.hp.com/enterprise/cache/328211-0-0-224-121.html.
135 See JBoss indemnification program at: http://www.jboss.com/pdf/press/indemnification0405.pdf.

software over FOSS alternatives because they felt the availability of specific pieces of software does not depend on whether the software is proprietary or open source. More specifically, companies considered that in the event some third parties' component or tool is hit by a claim, their situation would be the same with an open source or proprietary component. Some companies indicated that even if the use proprietary products, companies would be unable to replace specific products if they were to suddenly become unavailable. Companies in the software field tend to move on very quickly. In the event this situation occurs, both proprietary and open source software users will face the same challenges as the other.

One company considered the lack of warranties on FOSS products to be a cause for concern, but it is certainly not the only reason for disregarding open source software and opting for proprietary versions. The company maintained that there are risks associated both with FOSS and with proprietary software. However, depending on circumstances, there are several reasons for choosing one alternative or the other: the lack of warranties and indemnifications on the FOSS side is but one of those. Reasons for companies to opt for proprietary software could be to get some warranties, but also for maintenance and support. A company's architecture can also play an important role in their choice. For instance, if the company's system is so inflexible that it cannot change to incorporate FOSS software, they are more likely to opt for proprietary software. In summary, the interviewed company opined that there is not a straight answer to the question of whether the lack of warranties on FOSS is a risk or not..

5. Concluding Remarks and Future Research

The substantial investment made by proprietary software firms in open source indicates that the nature of competition in the software industry has radically changed during the past two decades. Not only it is evident that the largely volunteer software movement has altered the basic nature of the software industry, but also it appears clear that the FOSS phenomenon has undergone a significant transformation from its free software origins to a more mainstream, commercially viable form.

Open source developments have experienced so much success that proprietary companies now incorporate open source strategies into their protection models, and very successful open source projects have had business models created around them. Despite the success of open source software models, it is highly unlikely that the proprietary software will wither away and die. The recent incorporation of proprietary features within some of the most prominent open source companies is clear evidence that IP rights have not decreased in importance during the past twenty years. On the contrary, it appears more likely that the proprietary and open source models will continue to co-exist as they have for a long time.

However, what this analysis shows is a shift in the way companies employ these models in the software industry. Although it remains clear that open source and proprietary software models will remain distinct, the data provides evidence demonstrating that in the future software will not receive licenses at either extreme.

The reason for this trend is simple: none of the currently available protection mechanisms individually succeed as a stand alone source of protection for computer programs. Instead, a balance of several protection mechanisms, including strictly IP models and FOSS, can meet the specific, customized needs of individual companies.

The *hybrid* model carries a large amount of potential legal risks. By combining legal theory and empirical research, this paper has provided a comprehensive analysis of the 'core' legal challenges surrounding the implementation of the *hybrid* model in the software context, and the coping mechanisms companies can implement in order to navigate such risks. Specifically, the empirical study confirmed the existence of all the theoretically formulated legal risks in practice, even though some were considered more concrete than others. The empirical research showed that the most challenging risks are those associated with the open source licenses, the compliance and compatibility with the licensing terms, and the risks associated with the 'copyleft' clause being the most concrete. These risks applied to both large and small companies. The least risky problems for the interviewed companies were associated with the lack of warranties and indemnifications on the open source side. Generally, the companies considered this feature equally problematic for closed and open source software. The problems associated with the ownership of rights were issues where companies did not have strong opinions and had not given it much consideration. The infringement risk was reported quite heterogeneously: even though all the companies agreed that incorporating open source into their final products does not enhance or reduce the risks of patent litigation, they provided different opinions on the reasons and consequences of the infringement and litigation risks (e.g. how widely the OSS package is used, field of technology, jurisdiction, etc...). Finally, only one company released software under FOSS licenses and, was the only participant that could comment on the dilution of the company's own intellectual property rights. The company was confident that the risk is minimal provided there is a thorough review of both the code and the FOSS licensing terms internally before releasing the final product.

The study showed that none of these methods is fully foolproof *per se*. Due to uncertainties surrounding the FOSS environment and the interpretation of the FOSS licenses, a well structured internal procedure and a solid design of the architecture and principles surrounding the overall model the company plans to incorporate is essential to reducing the legal challenges. Accordingly, all the interviewed companies conduct legal reviews and checks, either through an internal structured procedure or by delegating such responsibilities to one (or several) in-house staff members.

Generally, companies did not welcome the idea of a legislative entity to solve these problems, mainly due to the too slow legislative process in relation to the fast trends of the software industry. On the other hand, however, most companies felt the need for more case law and court interpretations.

The specific field of research at the centre of this essay has not been extensively investigated in previous literature. As mentioned earlier, even though several studies on the issue of intellectual property and open source software exist, they mostly

focus on FOSS as a phenomenon or proprietary software individually, and neglect to investigate the relationship and interaction between the two, the ways open source software affects companies' IP policies, and vice versa. Consequently, this case study relies on a number of theoretical assumptions formulated in the first part of the paper. Reliance on hypothesis was further enhanced by the still 'grey' area surrounding the interpretation of the FOSS licenses terms. One of the main difficulties in researching areas with ill-defined legal principles is the lack of clear-cut interpretations of the norms at hand. It is not realistic to claim that the findings of this study are applicable to all the companies that implement *hybrid* types of protection models. This was not the original purpose of the study nor does it represent its main contribution. Instead, the aim of the study (both the theoretical and the empirical part) was to generate new knowledge for strengthening the general understanding of the problems surrounding the use of *hybrid* models in the software field. The case study was relevant because it provided in-depth answers to the theoretical issues formulated in the first part of the paper. The empirical analyses shed light both on the most concrete legal risks associated with *hybrid* models, and on the coping mechanisms that are used to navigate such challenges.

The exercise conducted in the article was an explorative undertaking about the risks associated with *hybrid* protection mechanisms in the software industry. Specifically, this article limited itself to the legal problems surrounding the model. There is still much about software *hybrid* protection models that remains a mystery and warrants further research. Perhaps the more essential avenue for work in this area lies in the replication of similar studies among different participants to test the replicability of the results achieved. Another interesting avenue for research would be to explore different aspects of the open-closed business model in the software environment. For instance, the economic or managerial aspects of the problem could be investigated. The research conducted in this paper constitutes a solid basis for any further study that might aim at investigating or addressing problems related to the *hybridization* of other fields of technology. The biotechnology sector, for example, is another field where the closed-open innovation model is prominent but remains unexplored by researchers.

EKONOMI OCH SAMHÄLLE
Skrifter utgivna vid Svenska handelshögskolan

ECONOMICS AND SOCIETY
Publications of the Hanken School of Economics

213. ANU HELKKULA: Service Experience in an Innovation Context. Helsinki 2010.

214. OLLE SAMUELSON: IT-innovationer i svenska bygg- och fastighetssektorn. En studie av förekomst och utveckling av IT under ett decennium. Helsingfors 2010.

215. JOANNA BETH SINCLAIR: A Story about a Message That was a Story. Message Form and Its Implications to Knowledge Flow. Helsinki 2010.

216. TANJA VILÉN: Being in Between. An Ethnographic Study of Opera and Dialogical Identity Construction. Helsinki 2010.

217. ASHIM KUMAR KAR: Sustainability and Mission Drift in Microfinance. Empirical Studies on Mutual Exclusion of Double Bottom Lines. Helsinki 2010.

218. HERTTA NIEMI: Managing in the "Golden Cage". An Ethnographic Study of Work, Management and Gender in Parliamentary Administration. Helsinki 2010.

219. LINDA GERKMAN: Topics in Spatial Econometrics. With Applications to House Prices. Helsinki 2010.

220. IHSAN ULLAH BADSHAH: Modeling and Forecasting Implied Volatility. Implications for Trading, Pricing, and Risk Management. Helsinki 2010.

221. HENRIK HÖGLUND: Detecting Earnings Management Using Neural Networks. Helsinki 2010.

222. KATARINA HELLÉN: A Continuation of the Happiness Success Story: Does Happiness Impact Service Quality? Helsinki 2010.

223. MARIA JAKUBIK (Maria Jakubikne Toth): Becoming to Know: Essays on Extended Epistemology of Knowledge Creation. Helsinki 2011.

224. JOHANNA GUMMERUS: Customer Value in E-Service. Conceptual Foundation and Empirical Evidence. Helsinki 2011.

225. MARIA SOLITANDER: When Sharing Becomes a Liability: An Intellectual Capital Approach to Describing the Dichotomy of Knowledge Protection versus Sharing in Intra- and Interorganizational Relationships. Helsinki 2011.

226. BEATA SEGERCRANTZ: "... The Walls Fell Down but the Blokes Just Coded...". Varieties of Stability in Software Product Development during Organizational Restructurings. Helsinki 2011.

227. ERIC BREIT: On the Discursive Construction of Corruption: A Critical Analysis of Media Texts. Helsinki 2011.

228. MICHAEL WAGNER: Inventory Routing. A Strategic Management Accounting Perspective. Helsinki 2011.

229. NIKODEMUS SOLITANDER: Designing Creativity Through Clusters: A Periodisation of Cluster Discourse. Helsinki 2011.

230. ANDREAS PERSSON: Profitable Customer Management: a Study in Retail Banking. Helsinki 2011.

231. MATHIAS HÖGLUND: Microfoundations of HRM Effects: Individual and Collective Attitudes and Performance. Helsinki 2011.

232. ANETTE SÖDERQVIST: Opportunity Exploration and Exploitation in International New Ventures. A Study of Relationships' Involvement in Early Entrepreneurial and Internationalisation Events. Helsinki 2011.

233. OSKARI LEHTONEN: An Examination of How Entrepreneurs Can Improve Their Position in Relationship to Investors. Helsinki 2011.

234. IMOH ANTAI: Operationalizing Supply Chain vs. Supply Chain Competition. Helsinki 2011.

235. DMITRI MELKUMOV: Towards Explaining the Tasks and Roles of the Boards of Directors: The Role of Contextual, Behavioural and Social Identification Factors. Helsinki 2011.

236. CHARLOTTA NIEMISTÖ: Work/Family Reconciliation: Corporate Management, Family Policies, and Gender Equality in the Finnish Context. Helsinki 2011.

237. PAUL VIIO: Strategic Sales Process Adaptation: Relationship Orientation of the Sales Process in a Business-to-Business Context. Helsinki 2011.

238. KHALID BHATTI: Factors Affecting Knowledge Sharing in Strategic Alliances: The Role of Knowledge Sharing as Strategic Control Behavior among Multinational Enterprises. Helsinki 2011.

239. STEFAN GRANQVIST: Effektutvärdering inom företagandeutbildning. Helsingfors 2011.

240. HEIKKI RANNIKKO: Early Development of New Technology-Based Firms. A Longitudinal Analysis of New Technology-Based Firms' Development from Population Level and Firm Level Perspectives. Helsinki 2012.

241. PAULINA JUNNI: Knowledge Transfer in Acquisitions: A Socio-Cultural Perspective. Helsinki 2012.

242. HENRIKA FRANCK: Ethics in Strategic Management: An Inquiry into Otherness of a Strategy Process. Helsinki 2012.

243. SEPPO LAUKKANEN: Making Sense of Ambidexterity. A Process View of the Renewing Effects of Innovation Activities in a Multinational Enterprise. Helsinki 2012.

244. MARKUS WARTIOVAARA: Values and Freedom: An Inquiry into the Rise and Fall of Billionaire Wealth. Helsinki 2012.

245. SAINT KUTTU: Essays on Volatility and Time Varying Conditional Jumps in Thinly Traded African Financial Markets. Helsinki 2012.