

The London School of Economics and Political Science

Authoring Collaborative Projects: A Study of Intellectual Property and Free and Open Source Software (FOSS) Licensing Schemes from a Relational Contract Perspective

Chenwei Zhu

Submitted to the London School of Economics and Political Science for the degree of Doctor of Philosophy

London, October 2011

Declaration

I certify that the thesis I have presented for examination for the PhD degree of the London School of Economics and Political Science is solely my own work.

The copyright of this thesis rests with the author. Quotation from it is permitted, provided that full acknowledgement is made. This thesis may not be reproduced without the prior written consent of the author.

I warrant that this authorisation does not, to the best of my belief, infringe the rights of any third party.

Abstract

The emergence of free and open source software (FOSS) has posed many challenges to the mainstream proprietary software production model. This dissertation endeavours to address these challenges through tackling the following legal problem: how does FOSS licensing articulate a legal language of software freedom in support of large-scale collaboration among FOSS programmers who have to face a rather hostile legal environment underlined by a dominant ideology of possessive individualism? I approach this problem from three aspects. The first aspect examines the unique *historical* context from which FOSS licensing has emerged. It focuses on the most prominent “copyleft” licence—GNU General Public Licence—which has been shaped by the tension between the MIT-style hacker custom and intellectual property law since the 1980s. The second aspect tackles the *legal* mechanism of FOSS licences, which seems not dissimilar from other non-negotiated standard-form contracts. My analysis shows that FOSS licences do not fit well with the neoclassical contract model that has dominated software licensing jurisprudence so far. I therefore call for replacing the neoclassical approach with Ian Macneil’s Relational Contract Theory, which has remained conspicuously absent in the software licensing literature. The third aspect explores FOSS programmers’ *authorship* as manifested in FOSS licensing. It argues that the success of a FOSS project does not merely depend on the virtuosity of individual programmers in isolation. More importantly, a core team of lead programmers’ efforts are essential to channel individual authors’ virtuosity into a coherent work of collective authorship, which can deserve credit for the project as a whole. The study of these three aspects together aims to create a synergy to show that it is possible to graft a few collaborative elements onto the existing legal system—underpinned by a neoliberal ideology assuming that human beings are selfish utility-maximising agents—through carefully crafted licensing schemes.

Acknowledgements

I am deeply grateful to be given the opportunity to study at the LSE, where I have spent a few most memorable years focusing on a subject that I am fascinated about.

I am most thankful to my lead supervisor Ms. Anne Barron who has been tremendously supportive and patient in shepherding my academic project over these years. It is under Anne's careful guidance that I am able to channel my initial curiosity into this final doctoral work.

I express my heart-felt gratitude to Professor Linda Mulcahy who helped me enormously on Relational Contract Theory, which turned out to be such a crucial component of the whole dissertation. Linda's encouragement and kindness greatly helped me to finish this dissertation during my final year of study.

I am also very thankful to Dr. Edgar Whitley who has given much support during my early years at the LSE and I have benefited considerably from his advice.

A large part of this dissertation was written during my residency at the Lilian Knowles House (a student hall in East London), where I spent significant amounts of time studying and writing in its basement computer room. While there, I also made a few good friends whose presence made my many long writing sessions less lonely. My final two chapters were finished when I was lodging with two families at two different times (first with Gwenda and Dafydd's and then Yin and Bob's). Both families have warmly treated me like a family member and made great efforts to provide a quiet place for me to concentrate and study during the final lap of my PhD journey. I also want to thank Carol Capel-Bradford, Bal Khela, Sebastian Szuhay, David Possee and Joanna Sedillo, each of whom has spared some time to read parts of this dissertation and I certainly benefit a lot from their helpful comments and feedback.

Finally, I do not know how to thank enough my parents, who have made many sacrifices for my education from my beginning stage until now. This dissertation is dedicated to them.

C.W.

CONTENTS	
DECLARATION	2
ABSTRACT	3
ACKNOWLEDGEMENTS	4
LIST OF CASES	8
CHAPTER 1 OVERVIEW: PROBLEMATISING FOSS LICENSING	10
1.1 INTRODUCTION	10
<i>1.1.1 Two Conflicting Traditions: Where Do FOSS Licences Come From?</i>	<i>11</i>
<i>1.1.2 Three Aspects of FOSS Licensing: Framing the Questions</i>	<i>13</i>
1.2 KEY CONCEPTS IN FOSS LICENSING	15
<i>1.2.1 Source Code and FOSS</i>	<i>16</i>
<i>1.2.2 “Free Software” and “Open Source”</i>	<i>17</i>
<i>1.2.3 FOSS Stewardship and the Hacker Ethic</i>	<i>19</i>
<i>1.2.4 FOSS Licence and “Copyleft”</i>	<i>21</i>
1.3 STEWARDING FOSS PROJECTS: WHAT LICENCES CAN AND CANNOT DO	24
<i>1.3.1 Collaboration in FOSS Projects</i>	<i>24</i>
<i>1.3.2 The Role of FOSS Licensing</i>	<i>31</i>
1.4 STRUCTURE OF THE DISSERTATION	40
CHAPTER 2 FROM THE HACKER ETHIC TO “OPEN SOURCE”: A BRIEF HISTORY. 42	
2.1 INTRODUCTION: THREE HISTORICAL STAGES	42
2.2 FROM THE 1950S TO THE EARLY 1980S: THE PRE-LICENSING ERA	44
<i>2.2.1 Beginning of the Hacker Ethic</i>	<i>44</i>
<i>2.2.2 Decline of the Hacker Ethic</i>	<i>50</i>

2.3 FROM THE EARLY 1980S TO 1998: CLASH BETWEEN THE TWO TRADITIONS.....	54
2.3.1 <i>Changes in Market and Law</i>	54
2.3.2 <i>The Birth of Copyleft</i>	58
2.4 FROM 1998 ONWARDS: CHALLENGE FROM “OPEN SOURCE”	68
2.5 CONCLUSION.....	80
CHAPTER 3 INTELLECTUAL PROPERTY AND SOFTWARE FREEDOM	82
3.1 INTRODUCTION	82
3.2 “INTELLECTUAL PROPERTY” AND FOSS	83
3.3 COPYRIGHT AND FOSS	86
3.3.1 <i>The Originality Threshold</i>	87
3.3.2 <i>Software as Expression and Function</i>	92
3.3.3 <i>Scope of Exclusivity: Restricted and Permitted Acts</i>	98
3.4 PATENT AND FOSS	102
3.4.1 <i>Patentability of Software-Related Inventions</i>	103
3.4.2 <i>Perceived Threat of Patents to Software Innovation</i>	112
3.5 GPL AND SOFTWARE FREEDOM	117
3.5.1 <i>GPL as a Copyright and “Copyleft” Licence</i>	117
3.5.2 <i>GPL as a Patent Licence and its Limit</i>	126
3.6 CONCLUSION.....	128
CHAPTER 4 UNDERSTANDING FOSS LICENCES AS STANDARD FORMS—A	
RELATIONAL CONTRACT PERSPECTIVE	130
4.1 INTRODUCTION	130
4.2 FOSS COLLABORATION: DISCRETE TRANSACTION OR RELATIONAL CONTRACT?.....	132
4.2.1 <i>Discretist Approach: “Presentation” of Total Obligation</i>	133
4.2.2 <i>Relational Approach: Projecting Exchange into the Future</i>	138
4.3 THREE DOCTRINAL ROUTES TO ENFORCING A FOSS LICENCE	146
4.3.1 <i>First Route: Contractual Licence</i>	147
4.3.2 <i>Second Route: Bare Licence</i>	159
4.3.3 <i>Third Route: Promissory Estoppel</i>	164

4.4 CONCEPTUALISING THE GPL AS A RELATIONAL CONTRACT.....	166
4.4.1 <i>Two Obstacles: Classical and Neoclassical Laws</i>	166
4.4.2 <i>GPL as an Umbrella Agreement: Balancing Flexibility with Certainty</i>	173
4.5 CONCLUSION.....	178
CHAPTER 5 THE IDEA OF AUTHORSHIP IN FOSS LICENSING	180
5.1 INTRODUCTION	180
5.2 INDIVIDUAL AND COLLECTIVE “AUTHORS” IN FOSS PROGRAMMING.....	183
5.2.1 <i>Debating the Legacy of Romantic Aesthetics</i>	183
5.2.2 <i>Programming as an Engineering Discipline: Questioning “Originality”</i>	187
5.2.3 <i>Stewarding a FOSS Project: Questioning “Individuality”</i>	189
5.3 DEVELOPMENT OF THE LEGAL PERSONA OF FOSS PROGRAMMERS.....	200
5.3.1 <i>Claiming FOSS Authorship under Law (I): Copyright</i>	202
5.3.2 <i>Claiming FOSS Authorship under Law (II): Trademark</i>	214
5.3.3 <i>Legal Persona of Author-Stewardship</i>	220
5.4 CONCLUSION.....	227
CHAPTER 6 CONCLUSION	228
6.1 CONTRIBUTIONS TO THE SCHOLARLY LITERATURE.....	228
6.2 AVENUES FOR FUTURE RESEARCH	237
6.3 CONCLUDING REMARKS.....	244
BIBLIOGRAPHY	245
APPENDIX (A): DEVELOPMENT OF “INTELLECTUAL PROPERTY” AND FOSS: A	
TIMELINE	258
APPENDIX (B): GNU EMACS GENERAL PUBLIC LICENSE (1985).....	260

List of Cases

Aerotel Ltd. v. Telco Holdings Ltd. [2007] 7 RPC 117

Apple Computer, Inc. v. Microsoft Corporation, 35 F.3d 1435 (9th Cir. 1994)

Arizona Retail v. Software Link, 831 F. Supp. 759 (D.Ariz. 1993)

Baird Textile Holding Ltd. v. Marks & Spencer plc. [2001] EWCA Civ 274

Beta Computers (Europe) v. Adobe Systems (Europe) (1996) SLT 604

Bilski v. Kappos, 130 S. Ct. 3218 (2010)

British Leyland Motor Corpn v. Armstrong Patents Co Ltd. [1986] AC 577

Cantor Fitzgerald International v. Tradition (UK) Ltd. [2000] RPC 95

Computer Associates International, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir.1992)

Currie v. Misa (1975) L.R. 10 Ex. 153

Diamond v. Diehr, 450 U.S. 175 (1981)

Feist Publication Inc. v. Rural Telephone Service Inc., 499 U.S. 340 (1991)

Follett v. New American Library 497 F. Supp. 304 (SDNY, 1980)

Gates Rubber v. Bando Chemical Ltd., 9 F.3d 823 (10th Cir. 1993)

Gilliam v. ABC, 538 F.2d 14 (2d Cir.1976)

Gottschalk v. Benson, 409 U.S. 63 (1972)

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

Jacobsen v. Katzer, 535 F.3d 1373 (Fed. Cir. 2008)

Lotus Development Corp. v. Paperback Software International, 740 F Supp 37 (D Mass, 1990)

Lotus Development Corp. v. Borland International, 49 F.3d 807 (1st Cir. 1992); 516 US 233(1996)

Microsystems Software, Inc. v. Scandinavia Online AB, 98 F. Supp. 2d 74 (D.Mass., 2000), aff'd, 226F. 3d 35(1st Cir., 2000)

Planetary Motion v. Techsplosion, 261 F.3d 1188 (11th Cir.2001)

Pollstar v. Gigmania, Ltd., 170 F.Supp. 2d 974 (E.D. Cal. 2000)

ProCD v. Zeidenberg, 86 F.3d 1447 (7th Cir.1996)

Richardson v. Flanders [1993] FSR 497

Saphena v. Allied Collection [1995] FSR 616

Specht v. Netscape Communications Corp., 306 F.3d 17 (2d Cir. 2002)

State Street Bank v. Signature Financial Group, 149 F. 3d 1368 (Fed. Cir. 1998)

Step-Saver Data Sys. Inc. v. Wyse Tech, 939 F.2d 91 (3rd Cir. 1991)

Symbian Ltd. v. Comptroller-General of Patents [2008] Bus. L.R. 607

Ticketmaster Corp. v. Tickets.com, Inc., U.S. Dist. LEXIS 6483 (C.D. Cal. 2003)

University of London Press Ltd. v. University Tutorial Press Ltd. [1916] 2 Ch 601

Vicom/Computer-related Invention, T208/84 [1987] EPOR 74; [1987] OJ EPO 14

Whelan Associates Inc. v. Jaslow Dental Laboratory Inc., 797 F.2d 1222 (3d Cir. 1986)

Chapter 1 Overview: Problematising FOSS Licensing

1.1 Introduction

The emergence of free and open source software (FOSS) development as a method for producing highly robust information products, such as the Linux operating system or the Apache web server, has challenged many conventional ideas that underpin the business model of proprietary software.¹ What is truly remarkable about these FOSS projects is their ability to attract a wide-range of voluntary contributors across the globe, whose contributions are produced largely without immediate monetary incentives.² Benkler calls this phenomenon networked “peer production”, where innovation is decentralised to its maximum and creative individuals follow neither price signals under the market mechanism nor managerial commands within a hierarchical corporate structure.³ In fact, the “peer production” model goes beyond FOSS and it has already inspired many non-programming creative activities to be conducted on the mass collaborative level in a similar way.⁴

Most significantly, each FOSS project can be seen as generating a software commons, in which source code is freely accessed, used, modified and redistributed under

¹ Eric Raymond, *The Cathedral and the Bazaar*, 2000, Version 3.0 at <<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>> (hereafter *Cathedral*)

² For a survey of the motivational forces behind FOSS contribution, see Karim R. Lakhani and Robert G. Wolf, “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects”, in *Perspective on Free and Open Source Software*, eds. by Feller, Fitzgerald, Hissam & Lakhani (Cambridge, Mass.: MIT Press, 2005)

³ Benkler finds that the production model of FOSS operate largely outside the firm-based or market-based structure: “Free software projects do not rely either on markets or on managerial hierarchies to organize production. Programmers do not generally participate in a project because someone who is their boss instructed them, though some do. They do not generally participate in a project because someone offers them a price, though some participants do focus on long-term appropriation through money-oriented activities, like consulting or service contracts. But the critical mass of participation in projects cannot be explained by the direct presence of a command, a price, or even a future monetary return particularly in the all-important microlevel decisions regarding selection of projects to which participants contribute.” See Yochai Benkler, “Coase's Penguin, or, Linux and ‘The Nature of the Firm’” (2002) 112, (3) *Yale Law Journal* 369, at 372-3

⁴ Wikipedia is a glaring example here. Richard Stallman sees Wikipedia as a natural extension of FOSS collaboration into the area of encyclopaedias. See Stallman, “The Free Universal Encyclopedia and Learning Resource” at <<http://www.gnu.org/encyclopedia/free-encyclopedia.html>>; see also Don Tapscott, and Anthony D. Williams, *Wikinomics* (London: Portfolio, 2006); Charles Leadbeater, *We-Think* (London: Profile Books, 2008)

certain rules specified in a corresponding FOSS licence⁵. It is important to note that a FOSS licence does not make programmers entirely abandon their intellectual property rights altogether into the public domain, but it carefully retains some private ownership rights for the purpose of preserving and nurturing the software commons.⁶

1.1.1 Two Conflicting Traditions: Where Do FOSS Licences Come From?

One of the most interesting but also puzzling issues that concern this dissertation is a paradox as manifested in the software commons created by collaborative FOSS projects. There seem to be two conflicting notions that are welded together in these commons-oriented regimes: 1) programmers' "stewardship" responsibility to share software with the public, and to develop it collaboratively and 2) their individual "private property" rights in the software code that is produced. These two notions stem respectively from the two almost diametrically opposed traditions of producing and circulating software. In this dissertation, I call the first one the "stewardship" tradition, where software is widely shared in the community⁷, and the second one the "private property" tradition, where exclusive property rights in software are held by its authors. The two traditions have different pedigrees. The first tradition of software stewardship is derived from the computer hacker culture originated in the 1950s and 1960s in some leading US computer research labs such as the MIT Artificial Intelligence (AI) Lab, while the second tradition is institutionalised in intellectual property law (especially copyright) that started to cover software as a subject matter in the early 1980s.⁸

⁵ A FOSS licence is sometimes seen as the constitution of the corresponding software-sharing community. Weber comments that FOSS licences can be read as the statement giving "constitutional message" to a community and it should reassure that all programmers "will be treated fairly if they join the community." Steven Weber, *The Success of Open Source* (Cambridge, Mass.: Harvard Uni. Press, 2004) *Success*, p.179 (Hereafter *Success*)

⁶ FOSS projects are not embodiment of anarchy, but there is organisation and governance. It is argued that stewardship is an important mode of governing the software commons enabled by FOSS licences. See Chris DiBona, Danese Cooper, and Mark Stone, "Introduction", in *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O'Reilly, 2006) p. xxxvii

⁷ A few leading FOSS programmers calls for adopting the term "stewardship", which is believed to be a more accurate term in describing FOSS practice in managing software commons. See, for example, Chris DiBona, Danese Cooper, and Mark Stone, "Introduction" to *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O'Reilly, 2006) p. xxxvii

⁸ The US Congress amended their 1976 Copyright Act in 1980 and put "software" under the protection the same way that "literary works" are protected. The similar thing happened in the UK in the early 1980s as well.

The hacker's stewardship tradition follows the so-called "Hacker Ethic", which was first documented in the form of six tenets by Steven Levy in his famous book—*Hackers: Heroes of the Computer Revolution*—published in 1984.⁹ From the 1950s to the early 1970s, software was shared among computer hackers and it was impossible for anyone to claim exclusive ownership rights in software because copyright was not established enough to include software as a subject matter. This Hacker Ethic is important because it is said to have a lasting impact on the "shared identity and belief system" of today's FOSS programmers.¹⁰ It has also seeded an important norm of collaboration for any sustainable FOSS project, where software code should be shared as widely as possible and there should be no barrier to artificially block the flow of information¹¹.

However, in the late 1970s, this hackers' stewardship tradition of information sharing began to be eroded by the rise of proprietary software. There were two developments that contributed to this erosion. First, software came under trade secrecy protection. Many hackers were required to sign non-disclosure agreements when they were lured away to write code for proprietary software companies. Secondly, legislation was changed to make copyright subsist in software. In 1980, the US Congress extended its copyright law to explicitly cover software programs¹². Copyright later became a main mode of IP that grants programmers' exclusive ownership in software.¹³

The FOSS licences came into existence exactly during this historical context where the old Hacker Ethic came into intense conflict with the new trend of owning proprietary rights in software. In response to the ascendancy of proprietary software, some hackers began to experiment with the idea of crafting copyright licences to specify the programmers' stewardship responsibility of software sharing. The most prominent example is the GNU General Public License (GPL), which was designed

⁹ Hereafter Levy, *Hackers* (London: Penguin Books, 1984,1994)

¹⁰ Steven Weber, *Success*, supra note 5, p.144

¹¹ For example, the second tenet of the Levy's ethic says that "all information should be free", which later becomes an important, though not entirely uncontroversial, norm in the internet age.

¹² 17 U.S.C. s.101

¹³ Specifically, the US Copyright Act gives copyright owners including software programmers the "exclusive right" to do certain activities, and it would be illegal for non-owners to do these activities without permission.17 USC s. 106; However, it should not be forgotten that the copyright owners' "exclusive right" have two exceptions in software: First, in order to run the software program, it should be allowed to copied to hard disk and computer's memory; second, users are allowed to make back-up copies. 17 U.S.C. s.117

and has been perfected by an ex-MIT hacker Richard Stallman since the 1980s.¹⁴ The GPL is the first and most widely adopted licence among FOSS developers. It contains an innovative feature known as a “copyleft” clause that enjoins the downstream developers to share their modifications and improvements of the GPL covered code.¹⁵ Copyleft is especially useful for those community-based projects such as the Linux kernel to grow and expand outside the market mechanism and the hierarchical corporate structure. It is interesting to note that copyleft licences, including the GPL, do not dispense with copyright. The paradox is that their imposition of the “share-alike” responsibility in copyleft is dependent upon the broad property rights granted by the copyright regime in the first place. In order to make sense of this paradox, it should be borne in mind that the GPL (and other FOSS licences) is an attempt to reconcile two antagonistic traditions battling to gain influence over the way that software is produced and distributed. In other words, copyleft is the computer hackers’ legal experiment to graft the old hacker culture onto the IP law system through the device of FOSS licensing. Given the hugely complex and paradoxical nature of the subject, I need to further narrow the dissertation down to three more specific aspects of FOSS licensing and its role in FOSS collaboration in the following sub-section.

1.1.2 Three Aspects of FOSS Licensing: Framing the Questions

The main thrust of this dissertation is to study collaborative relations through the lens of FOSS licensing, which is shaped by the tension between the tradition of stewardship and that of private ownership. With this tension firmly in the background, I frame the research questions under three interrelated aspects of FOSS licensing:

- **Historical Aspect** The first aspect tackles the question as to the origin the FOSS stewardship tradition and how it has managed to coordinate large-scale

¹⁴ Stallman wrote the very first copyleft licence known as the Emacs General Public Licence (EGPL) in 1985. It was a licence specifically designed for the Emacs programming editor. It was been amended a couple times before it finally was turned into the generic GPL 1.0 in 1989. The 1985 EGPL licence was the solo work of Stallman intended to retain the Emacs culture of software sharing, which nonetheless became increasingly vulnerable facing the rise of proprietary software. I will give a more detailed account of the birth of copyleft in Chapter 2.

¹⁵ s. 2(b) GPL v2.0; s. 5(c) GPL v.3.0

collaboration among programmers. I will show the historical context where computer hackers' collaborative ethos was first gestated and later concretised into software stewardship responsibility as detailed by FOSS licences. I define "stewardship" as FOSS programmers' responsibility to preserve and protect the *commons* where software is freely accessible, modifiable and redistributable. I further narrow down stewardship responsibility in this dissertation to the software developers' specific duty to share software pursuant to 1) the Hacker Ethic (documented by Steven Levy in 1984) 2) Free Software Definition (by Stallman) and 3) Open Source Definition (by Raymond and Perens). I will show, through the lens of FOSS licensing, why the "commons" held in stewardship is critical to the success of any large-scale "peer produced" collaborative software project.

● **Legal Aspect (Intellectual Property and Contract)** The second aspect explores the jurisprudence of FOSS licensing schemes which covers both (intellectual) property and contract laws. It tries to tackle the question as to how FOSS licensing has attempted to graft FOSS stewardship responsibility (to secure software freedom) onto the IP system. I find that the existing doctrinal rules from IP and contract laws, which assume that economically minded individuals compete against each other in zero-sum games, do not satisfactorily explain the highly collaborative relation that FOSS licences intend to support. Instead, I will employ Ian Macneil's Relational Contract Theory (RCT)¹⁶ to shed some new light on the issue. I argue that licences used by any successful FOSS projects are actually a kind of relational contract involving a high degree of cooperation over a long period of time, rather than a series of one-shot discrete transactions.

● **Authorial Aspect** The third aspect asks: who are the "authors" of FOSS? How is programmers' authorship manifested in FOSS licences? What motivates authors to contribute to software commons in a seemingly altruistic manner? I will show that the highly collaborative nature of FOSS authorship hardly conforms to the Romantic vision of the authors as solitary individual geniuses that are assumed by the orthodox IP legal institution. The practice of FOSS licensing reflects FOSS programmers' desire to be credited as authors of their

¹⁶ I.R. Macneil, *The New Social Contract—An Inquiry into Modern Contractual Relations* (New Haven and London: Yale University Press, 1980)

contributions. Most importantly, FOSS authors are not just individual creators of code in isolation. When individually created contributions are pieced together into a whole, the coordinating efforts behind the project would give birth to a collective authorship that can be held responsible and deserve credit for the production of the FOSS project as a whole. This collective authorship is closely related to the lead programmers' stewardship responsibility to forge collaboration among individual programmers driven by a multiplicity of motivational forces. It is exactly this alignment of authorship with stewardship that has fundamentally challenged the author-ownership model that dominates the conventional IP jurisprudence.

The study of these three aspects together will create a synergy to show that FOSS programmers' struggle to rebuild some elements of a collaborative ethos that originated from the old hacker ethic but has been eclipsed by the rise of intellectual property (especially copyright) regulation of software innovation. This struggle runs against a dominant neo-liberal understanding of modern property and contract institutions as mainly furthering the economic interests of atomised individuals in isolation. It results in programmers' minimum stewardship responsibility being verbalised into FOSS licensing terms, which becomes the legal infrastructure that large-scale collaboration can rely upon. Although the role of FOSS licensing in facilitating collaboration is important, my thesis by no means intends to exaggerate this role. FOSS licensing alone does not make collaboration happen, but it must be combined with other non-legal decisions made by programmers' one integrated project in a radically decentralised environment¹⁷. I will start by clarifying a few basic key concepts that will help to understand FOSS programmers' licensing schemes in relation to their collaborative efforts.

1.2 Key Concepts in FOSS Licensing

This section is written to clarify a few of the most basic concepts that are of great importance to this dissertation. I will divide them into four groups, each of which deals with two closely related concepts in pair. I will give each concept a concise

¹⁷ One of these non-legal factors is the technical decision made by lead programmers to create a modular architecture, where software can be modified by many collaborating programmers. See Section 1.3.1 of this chapter for more detail.

definition and then explain briefly their importance in the context of FOSS licensing. I believe these explanations will facilitate the understanding of the subject when the dissertation progresses into a more detailed and technical discussion.

1.2.1 Source Code and FOSS

Source code is the technical term used to describe the human-readable code that is written by programmers. Not unlike other human language, source code is written in the alphanumeric form, which can be understood or possibly altered by other programmers. Source code needs to be turned into machine-readable **object code** through a compiler program before it can be run by a computer. This process is known as “compilation”.¹⁸

The significance of source code is threefold. First, because source code can be written and read by human beings, it is not drastically dissimilar from literary text. For this reason, most countries make software eligible for copyright protection under the category of literary work by analogy¹⁹. Second, software can be easily modified through changing the source code and thus opens up the possibility for other programmers to make adaptations to their own needs or improve the software collaboratively. Thirdly, because the source code can be read by human beings, it makes software not only a technological artifact but also a *communicative process*. That’s why software can also be seen as a kind of “discourse”.²⁰ This “communicative” or “discursive” feature has a broader social consequence. It leads the anthropologist Christopher Kelty to believe that FOSS is a kind of public sphere where FOSS programmers argue not only “about” technology but also “through” technology (as if the source code is their human language).²¹ So by hiding the source code, software will be effectively deprived of its communicative potential, which goes against the original design of this technology.

¹⁸ David Bainbridge, *Legal Protection of Computer Software* (Heywards Heath, West Sussex: Tottel Publishing, 2008, 5th Ed.) p.57

¹⁹ For example, s.3(1), UK CDPA 1988

²⁰ Fitzgerald points out that software is kind of discourse due to its communicative nature: “Software in the information society is discourse. It is not simply a literary text (a copyright law categorisation) it is fundamental to communicative architecture.” Brian Fitzgerald, “Software as Discourse? The Challenge for Information Law” (2000) 22 (2) E.I.P.R. 47

²¹ Kelty, *Two Bits—The Cultural Significance of Free Software*, (Durham: Duke University Press, 2008), p.29 (Hereafter *Two Bits*)

FOSS can be defined as the kind of software whose source code is publicly available with no restriction on modification and redistribution of it. In contrast, proprietary “closed-source” software developers release software only with non-human readable object code without disclosing the corresponding source code. Stallman points out:

Source code is useful (at least potentially) to every user of a program. But most users are not allowed to have copies of the source code. Usually the source code for a proprietary program is kept secret by the owner, lest anybody else learn something from it. Users receive only the files of incomprehensible numbers that the computer will execute. This means that only the program’s owner can change the program.²²

It is important to know that disclosure of the source code by itself is not enough to qualify software as FOSS. We must look at the definitions of “free software” and “open source” for more detailed guide in the following sub-section.

1.2.2 “Free Software” and “Open Source”

Free software is more than just publicly disclosed source code. The Free Software Foundation (founded by Richard Stallman) publishes *The Free Software Definition* (FSD), defining **free software** as the type of software that gives its users four kinds of freedom:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.²³

²² Richard Stallman, “Why Software Should Be Free”, 1991, at <http://www.gnu.org/philosophy/shouldbefree.html>

²³ Richard Stallman, “The Free Software Definition” at <http://www.gnu.org/philosophy/free-sw.html>

Along the same line, Open Source Initiative (co-founded by Eric Raymond and Bruce Perens) publishes the *Open Source Definition* (OSD) including a long and detailed list comprising ten criteria. Perens distils the ten criteria into the three principles. It defines **open source** software as giving software users three kinds of rights:

- The right to make copies of the program, and distribute those copies.
- The right to have access to the software's source code, a necessary preliminary before you can change it
- The right to make improvements to the program.

Except that the wording is slightly different, the FSD and the OSD means almost the same thing in terms of the *duty* that programmers should bear: FOSS programmers should give users the “freedoms” (as in the FSD) or the “rights” (as in the OSD) to access, copy, modify and redistribute the software. Based on this reason, some scholars think that there is no pronounced difference between “free software” and “open source” because both labels describe the same type of technological artifact or the same type of programming practice.²⁴

Unfortunately, this view is *not* held by the people who have respectively authored the FSD and the OSD. There has been a long-standing schism between the two camps. Stallman on the side of the FSD, believes that free software campaigners and open source advocates hold different visions about the future of non-proprietary software. According to him, free software is a “social movement” to enlarge users’ software freedom, while open source is merely a software “development methodology” that claims itself to be superior to proprietary software.²⁵ Raymond, from the camp of “open source”, distances himself from Stallman’s deep scepticism about commercialisation of non-proprietary software. He criticises free software movement for being “very zealous and very anticommercial”.²⁶ For Raymond and his followers, “open source” should break into the mainstream software market and its success

²⁴ For example, Keltly is the champion of this view. See Keltly, *Two Bits*, p.100

²⁵ Stallman, “Why Open Source Misses the Point of Free Software” at <<http://www.gnu.org/philosophy/open-source-misses-the-point.html>>

²⁶ Raymond, Section 2 “The Varieties of Hacker Ideology” in *Homesteading the Noosphere*, 2002, at <<http://www.catb.org/~esr/writings/homesteading/homesteading/>>

depends upon the pragmatic approach rather than the closed ideology of “software freedom”.²⁷

1.2.3 FOSS Stewardship and the Hacker Ethic

FOSS stewardship means programmers’ duty or responsibility to preserve and protect the *commons* where software is freely accessible, modifiable and redistributable. FOSS stewardship must be pursuant to the software developers’ duty listed in *Free Software Definition* and/or the *Open Source Definition*.

The Hacker Ethic is historically the main source of FOSS stewardship duty to share software, and it later evolves into the FSD and the OSD. It first developed in the computer hacker community such as the MIT AI Lab since the 1950s and the 1960s. It originally means the hackers’ duty to share any information concerning computer technology and this happens in an era when IP law had not yet been used to cover software. In the beginning, the Hacker Ethic was largely a body of unwritten rules, and it was “an ethic seldom codified, but embodied instead in the behaviour of hackers themselves.”²⁸ In 1984, Steven Levy, in his highly regarded pioneering study of the hacker culture, identifies six widely recognised tenets of the Hacker Ethic. The first three tenets are the most relevant to the software stewardship obligation in this dissertation:

- Tenet 1: Access to computers—and anything which might teach you something about the way the world works—should be unlimited and total. Always yield to the Hands-on Imperative!
- Tenet 2: All information should be free.
- Tenet 3: Mistrust Authority—Promote Decentralisation.

²⁷ My position on this issue is that there is *both* consensus *and* division between the two camps and it would be wrong to see only one side of the story. The consensus and the division happen on two different levels. First, on the technical level, the two camps agree on the technical definition of non-proprietary software (in the FSD and OSD) and there is a consensus that software developers should have the same set of stewardship obligations to share software as listed in both the FSD and the OSD. Secondly, on the ideological level, free software campaigners and open source advocates disagree on the ideology behind their respective causes. Stallman’s “free software” is a belief in the intrinsic value of “freedom” as the ultimate driving force of the movement. Raymond’s “open source” does not wish to engage with the philosophical discourse of “freedom”, but it adopts a more pragmatic market-friendly approach. For more detail about difference between the two camps in a historical context, see Section 2.4, Chapter 2 of this dissertation for more detail.

²⁸ Levy, *Hackers*, p.7

The first two tenets calling for “unlimited and total” access to computers (Tenet 1) and “all information should be free” (Tenet 2) lays the ethical foundation for the FSD and the OSD, while Tenet 3 of “Mistrust Authority—Promote Decentralisation” anticipates the decentralised “peer production” model that marks the success of FOSS projects. Furthermore, there is also a difference between Levy’s ethic and the later FSD and OSD. The unlimited and total access to “computers” in the first tenet of the Hacker Ethic covers both hardware and software in its early days, whilst the FSD and the OSD is focused only on software, because the latter is intended to be the guideline for writing *software* licences.

Alternatively, the definition of the Hacker Ethic can also be found in the definitive *The New Hacker's Dictionary* (also known the “Jargon File” edited by Raymond). The dictionary defines the Hacker Ethic as the “belief that *information-sharing* is a powerful positive good, and that it is *an ethical duty* of hackers to share their expertise by writing open-source code and facilitating access to information and to computing resources wherever possible.”²⁹ (added emphasis) This is a good and succinct definition that encapsulated the core meaning of FOSS programmers’ stewardship duty which is adopted by this dissertation.

The old Hacker Ethic was later challenged and eroded by the rise of proprietary software in the late 1970s, but it was never fully defeated. A more recent study shows that the Hacker Ethic after the 1980s and until the beginning of the twenty-first century is still alive and well amongst FOSS programmers,³⁰ thanks to the advent of FOSS licences in the mid-1980s. These licences are written in the form of intellectual property licences to guard programmers’ core stewardship responsibility of software sharing against encroachment of proprietary software. This leads me to explain exactly what the FOSS licences are.

²⁹ According to the Jargon File, the Hacker Ethic has also a second meaning: “The belief that system-cracking for fun and exploration is ethically OK as long as the cracker commits no theft, vandalism, or breach of confidentiality.” This is not the ethic that is dealt with in this dissertation. “The Hacker Ethic” in Jargon File, compiled by Raymond, at <<http://www.catb.org/jargon/html/H/hacker-ethic.html>>

³⁰ Himanen’ study demonstrates the robustness of the Hacker Ethic in the information society. See Pekka Himanen, *The Hacker Ethic and the Spirit of the Information Age*, (NY: Random House, 2001)

1.2.4 FOSS Licence and “Copyleft”

A FOSS licence is defined as an “intellectual property” licence that gives software users the rights to access, copy, modify and redistribute the source code.³¹ The clauses in a FOSS licence should not impose any restriction that contradicts FOSS stewardship responsibility as detailed in the FSD and the OSD³². There are two types of FOSS licences. One is called copyleft licences and the other is known as permissive (non-copyleft) licence.

Copyleft Licences

A copyleft licence contains an anti-privatisation clause that enjoins downstream developer-users to share their publicly released modifications or improvements of the original software. In other words, any derivative works based on the original copylefted code, when publicly distributed, must be released under the same copyleft licence. In the mid-1980s Richard Stallman designed the first copyleft licence for his GNU Emacs programming editor. Later he turned this Emacs-specific licence into a generic template licence—GNU General Public License (GPL)—that can be used by any software.³³ The GPL is not only important to Stallman’s own GNU software project, but it is also crucial to the success of many other FOSS projects. The most

³¹ It is important to be aware that the umbrella term “intellectual property” (IP) normally comprises at least three main sub-areas: copyright, patent and trademark. Stallman points out that three sub-areas of are rather different and the term should be avoided whenever possible: “Copyright law was designed to promote authorship and art, and covers the details of expression of a work. Patent law was intended to promote the publication of useful ideas, at the price of giving the one who publishes an idea a temporary monopoly over it—a price that may be worth paying in some fields and not in others. Trademark law, by contrast, was not intended to promote any particular way of acting, but simply to enable buyers to know what they are buying. Legislators under the influence of the term “intellectual property”, however, have turned it into a scheme that provides incentives for advertising.” Stallman, “Did You Say ‘Intellectual Property’? It’s a Seductive Mirage” at <<http://www.gnu.org/philosophy/not-ipr.html>>

³² Larry Rosen distinguishes open source licences from proprietary licences against the criteria as to whether “software freedom” is protected or not. His definition is also helpful to illustrate the nature of a FOSS licence:

- “An *open source license* is the way a copyright and patent owner grants permission to others to use his intellectual property in such a way that *software freedom* is protected.”
- “A *proprietary licence* is the way a copyright and patent owner grants permission to others to use his intellectual property in a restricted way, through secrecy or other limitations, so that *software freedom* is not protected.”

Lawrence Rosen, *Open Source Licensing—Software Freedom and Intellectual Property Law* (Upper Saddle River, NJ: Prentice Hall PTR, 2005) p. 52

³³ For the historical context in which the GPL came out of the GNU Emacs dispute from 1983 to 1985, see Chapter 2 of this thesis and also Kelty, Chapter 6 “Writing Copyright Licences”, Two Bits, pp.179-209

successful one of them is no doubt the Linux kernel program that has been licensed under the GPL since its inception.

It is important to note that copyleft does not straightforwardly reverse or oppose copyright, but its legal mechanism relies on copyright. The preamble of the current version of the GPL (version 3.0) makes it clear that software developers use the GPL to protect users' rights with two steps: "(1) assert copyright on the software, and (2) offer [users] this License giving [users] legal permission to copy, distribute and/or modify it."³⁴ So copyright provides the basic legal framework for the software developers to enable software freedom in the first place. What is really innovative and central to copyleft is its "share-alike" clause, which is sometimes also known as the "viral" clause.³⁵ Section 5(c) of the GPL requires that the "modified source version" based upon the original GPL covered code must be licensed under the same GPL when publicly released:

You must license *the entire work*, as a whole, under this [GNU General Public] License to anyone who comes into possession of a copy. [...] This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.³⁶ (added emphasis)

Under this clause, if a follow-up programmer makes some changes to a piece of GPL covered code, then the modified source version as a whole (i.e. the "entire work" as in the clause), when publicly re-distributed, must be released under the same GPL. Copyleft is essentially an anti-privatisation device meticulously designed to prevent software programmers from hiding modifications of the GPLed code. It reconfigures the central function of copyright—which intends to give copyright owners *exclusive* control over their work—into an *anti-exclusionary* institution, where everything must be shared. In this way, GPLed software is made into an evolving object more than a static non-modifiable end-product and it is said to give its users an unbroken chain of

³⁴ Preamble, GNU GPL 3.0

³⁵ Andres Guadamuz, "Viral contracts or unenforceable documents? Contractual Validity of Copyleft Licenses", (2004) 26 (8) *European Intellectual Property Review* 331-339.

³⁶ Historically, the previous version of the GPL (v2.0) has been critical in the success of projects like the Linux project. Since the GPL v2.0 continues to be used until today, it is worth quoting its copyleft clause as well. Section 2(b) of the GPL v2.0 says: "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."

software freedom.³⁷ It is worth noting that the whole GPL licence is a much longer and more complicated document than this anti-privatisation copyleft clause. I will go back to discuss the GPL and its relation with intellectual property law in more detail in Chapter 3.

Permissive Licences (BSD-style Licences)

Not all FOSS licences are copylefted. There are non-copyleft licences as well. They are known as the “permissive licences”, which are sometimes also called BSD-style licences or “academic licence”³⁸ in the literature. Because permissive licences do not have copyleft’s anti-privatisation clause that forces the downstream developers to contribute modifications back to the community, it is more “permissive” than the GPL in this sense. Historically, permissive licences are associated with software distribution by academic institutions such as the University of California, Berkeley. For example, UC Berkeley publishes its own permissive licence called BSD (Berkeley Software Distribution) License, which is also widely adopted in the FOSS world. The BSD License is occasionally called “copycenter”, which indicates that it sits somewhere between copyright and copyleft.³⁹

Apart from the requirement of retaining the original copyright notice⁴⁰, the BSD licence allows the downstream users to do almost whatever they want in redistributing the source and object code. This means that in future distributions, the BSD licensed software is not obliged to be re-licensed under the same BSD licence. So it is possible for the original BSD licensed software to be released under other licences including proprietary licences. The direct upshot is that the initially freely available source code has the possibility to be privatised in future distributions. A notable example is Mac OS—Apple’s operating system—which contains a

³⁷ Free Software Foundation, “Rationale Document”, at <<http://gplv3.fsf.org/rationale>>

³⁸ Rosen, *supra* note 31, pp.73-74

³⁹ Eric Raymond *et. al.*, “Copycenter”, *The Jargon File* at <<http://catb.org/~esr/jargon/html/C/copycenter.html>>; Kirk McKusick, a computer scientist and a major contributor to the BSD system writes: “[...] Berkeley had what we called “copycenter,” which is “take it down to the copy center and make as many copies as you want.” You want to go off and do proprietary things with it? Fine, you can do that. You want to keep it out in the Open Source domain? You’re welcome to do that as well.” BSD Newsletter, “What is the BSD License?” at <<http://www.bsdnewsletter.com/bsd/license.html>>

⁴⁰ The BSD licence requires that redistribution of the source code and binary code “must retain the above copyright notice”. See “BSD License Template” at <<http://www.opensource.org/licenses/bsd-license.php>>

significant amount of FreeBSD code originally released under the BSD License. Apple modifies FreeBSD's code and then turns the modified version into proprietary software. Apple is allowed to do so because the BSD License, unlike the GPL, does not require downstream developers to disclose the source code of their modified versions.

1.3 Stewarding FOSS Projects: What Licences Can and Cannot Do

To steward a FOSS project is more than just to choose and use a plausible licensing scheme.⁴¹ A licence does not exist for its own sake, but its importance is realised through its being the legal expression of programmers' real collaborative experience that actually builds the FOSS project. In this dissertation, I argue that there are at least two elements that make a FOSS project sustainable for a lasting period of time: i) programmers' *collaboration* to integrate peer-produced contributions into a single coherent artefact and ii) a corresponding FOSS *licence* that is employed to facilitate this collaboration. The combination of the two elements makes the licence not merely a paper or electronic document on its own, but this licence is underpinned by a kind of "relational contract" with real lived collaborative experience among FOSS programmers. In particular, I will show that Ian Macneil's Relational Contract Theory (RCT) is helpful in analysing the FOSS projects' ability to engage long-term collaborative relations, which are sharply distinguished from the discrete commodity transaction model as assumed by proprietary software licensing practice. I will now explain the two elements.

1.3.1 Collaboration in FOSS Projects

The norm to "collaborate radically", according to Larry Sanger, is "one of the great innovations of the open source software movement."⁴² Here collaboration happens in

⁴¹ Tim O'Reilly, a prominent pro-FOSS entrepreneur, observes: "But open source is more than just a matter of licenses. Some of the most significant advances in computing, advances that are significantly shaping our economy and our future, are the product of a little-understood 'hacker culture.' It is essential to understand this culture and how it produces such innovative, high-quality software. What's more, companies large and small are struggling to understand how the ethic of free source code distribution affects the economic models underlying their present businesses." O'Reilly, "Lessons from Open-Source Software Development", (1999) 42 (4) Communications of the ACM 33 at 34

⁴² Sanger (the ex-chief architect of Wikipedia) means "radical collaboration" by the norm that "anyone can edit any part of anyone else's work". Radical collaboration is crucial to the success of

a radically decentralised environment and it is deeply rooted in the FOSS programmers' stewardship tradition where computers hackers address and solve almost all their technical problems collaboratively through information sharing since the 1950s and the 1960s.⁴³ Many later long-lasting FOSS projects owe exactly their success to this tradition of radical collaboration inherited from the older hacker community.

Radical collaboration may appear to be self-organised or spontaneous cooperative behaviour among FOSS contributors, but in fact it is *coordinated* by a small team of lead developers to make it happen.⁴⁴ In this light, I offer a refined definition of "radical collaboration" as identified by Sanger. I argue that "collaboration" in any successful FOSS project has two defining aspects: it is not only 1) *radically decentralised* but also 2) *coordinated* among a large number of contributors. First, what makes FOSS collaboration stand out is its radically decentralised structure capable of harnessing knowledge, intelligence and skills from potentially everyone with a minimum level of programming literacy connected by the internet. This radical openness allows individual innovators to delve into the tasks that truly pique their interest. It restores software programming activities as intellectual endeavours that are worth pursuing for their own sake.⁴⁵ Programmers satisfy their own curiosity in the process of exploring and solving technical problems instead of just following managerial commands in a firm, or monetary incentives on the market.⁴⁶ In short, a peer-production environment allows individuals to have a large degree of autonomy to follow their own intellectual pursuits under its radically decentralised and

Wikipedia, because it was "made possible for work to move forward on all fronts at the same time, to avoid the big bottleneck that is the individual authors, and to burnish articles on popular topics to a fine luster." Larry Sanger, "The Early History of Nupedia and Wikipedia: A Memoir", in *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O'Reilly, 2006) p.322

⁴³ See Section 2.2 of Chapter 2. See especially, Tenet 3 ("Mistrust Authority and Promote Decentralisation" of the Hacker Ethic documented by Levy.

⁴⁴ Although the vast number of non-core "peripheral" contributors' behaviours are rather more like self-organised spontaneous cooperation, these behaviours should not prevent us from seeing core developers' efforts to organise and coordinate collaboration among all contributors. For a detailed analysis of the "myth" of "self-organisation" in FOSS, see Weber, *Success*, supra note 5, pp.131-132; see also Marrella, Fabrizio & Yoo, Christopher S. "Is Open Source Software the New Lex Mercatoria?" (2007) 47 (4) *Virginia Journal of International Law*

⁴⁵ See Richard Sennett, *The Craftsman* (New Haven & London: Yale University Press, 2008) p.9, pp.24-25

⁴⁶ Raymond's famous aphorism that "every good work of software starts by scratching a developer's personal itch" captures this situation. Raymond, *Cathedral*, supra note 1

curiosity-driven structure.⁴⁷ Furthermore, it is also worth noting that collaboration is not unique to FOSS, and many corporate proprietary software projects also require some degree of collaboration. However, corporate collaboration happens on a much smaller scale and it is by no means radically decentralised, because innovation tends to be strictly restricted among the employed programmers and non-programming administrative staff within the compound of a particular company.⁴⁸

The second aspect of FOSS collaboration addresses an important weakness of radical decentralisation. No matter how innovative each individual programmer is, peer-produced contributions by themselves do not automatically integrate into one big piece of coherently functional software. The radical scale of decentralisation only adds tremendous difficulty to the task of integrating a heterogeneous amount of contributions into a whole. According to Surowicki, in a highly decentralised system, innovation becomes inevitably fragmentary and unsystematic because “there’s no guarantee that valuable information which is uncovered in one part of the [decentralised] system will find its way through the rest of the system” and [s]ometimes valuable information never gets disseminated, making it less useful than it otherwise would be.”⁴⁹

In order to compensate for this weakness of radical decentralisation, it is tremendously important for leaders of FOSS projects to *coordinate* many and varied peer-produced innovations. Or in Surowicki’s words, decentralised creations must be “aggregated” into a coherently functional whole. He is aware that “[a]ggregation—which could be seen as a curious form of centralization—is therefore paradoxically important to the success of decentralization.”⁵⁰ In fact, the more decentralised a system is, the more efforts are needed to aggregate peer-produced contributions

⁴⁷ See Benkler, Chapter 5 “Individual Freedom—Autonomy, Information, and Law” in *Wealth of Networks: How Social Production Transforms Markets and Freedom* (New Haven: Yale University Press, 2006) (Hereafter *Wealth*)

⁴⁸ Stallman points out that corporate proprietary software does not have radical openness as in FOSS collaboration. It strictly limits innovation within a closed corporate structure: “In any intellectual field, one can reach greater heights by standing on the shoulders of others. But that is no longer generally allowed in the [proprietary] software field—you can only stand on the shoulders of the other people *in your own company*.” (original emphasis) Stallman, “Why Software Should Be Free”, supra note 21

⁴⁹ James Surowicki, *The Wisdom of Crowds—Why the Many are Smarter than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*, (London: Little Brown, 2004) p.71

⁵⁰ *ibid.*, p75

together. Leaders of a FOSS project need to decide on (or sometimes speculate about) the level of decentralisation that they are willing and capable to cope with. In order to make a project both peer-productive and manageably aggregateable, a delicate balance must be drawn between decentralisation and aggregation. However, there is no fast and fixed rule about how this balance between the two should be kept. Situations vary from project to project. In order to find out how to make this balance, Raymond's essay *The Cathedral and the Bazaar* seems to give a clue to start with. Though an unflinching champion of decentralisation, Raymond recognises the importance of a "strong, attractive basic design" that is necessary to make a project aggregateable into one. This design is based on the two general pre-conditions that are needed for a decentralised Bazaar-style project to take off: "Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable time."⁵¹ In other words, a small group of core lead developers must go beyond the level of scratching their own programming itch, but they must also work hard to convince potential co-developers that their efforts are highly likely to be successfully aggregated "into something really neat in the foreseeable time." With these two pre-conditions in mind, I now need to delve a little deeper into the "strong, attractive basic design" mentioned by Raymond above, because it is crucial to the success of any collaborative FOSS project.

Designing a Collaborative Project

Not every kind of creative task is conducive to collaboration, let alone radically decentralised collaboration.⁵² A task that is radically collaborateable must have a modular architecture, which means it is capable of being divided into a lot of smaller improvable units and later pieced together into one coherent project. There are two parameters that matter here. One is the "modularity" and the other "granularity" of the task. Modularity concerns the extent to which a task "can be broken down into smaller components, or modules, that can be independently produced before they are

⁵¹ Raymond, *Cathedral*, supra note 1

⁵² For example, many traditional non-software creations, such as writing a novel or a academic paper, are done by a solo author or a very small number of collaborators, because these creative tasks are hard to be broken down into improvable fine-grained modules.

assembled into a whole,” while “granularity” concerns “the size of the modules, in terms of the time and effort that an individual must invest in producing them.”⁵³ Benkler observes that any “successful large-scale peer-production project must therefore have a predominate portion of its modules be relatively fine-grained.”⁵⁴ The fine-grained modularity that is conducive to peer-production collaboration is also often known as the “extensibility” of software among programmers.⁵⁵

Software projects are not by nature “extensible” with the right level of fine granularity, but they are *designed* to be so.⁵⁶ There are two types of design decision that leaders of FOSS projects have to make: one is the modular “architectural design” in software engineering terms and the other is the “legal design” for FOSS collaboration through FOSS licensing. The second type is exactly the main focus of this dissertation.

First, the architectural design concerns the software engineering problem of how to make a project “extensible” or modular at a manageable level. The GNU Emacs programming editor led by Stallman is a good example here. Stallman designs Emacs to be “extensible” in the sense that everyone can easily “go beyond simple customization and create entirely new commands” and these newly created commands “are simply programs written in the Lisp language, which are run by Emacs’s own Lisp interpreter.”⁵⁷ The Linux kernel project is another example of designed extensibility. Linus Torvalds, as the leader of the project, “followed good design principles, which allowed [Linux] to be extended in ways that he didn’t envision when he started work on the kernel.”⁵⁸ The modular architecture allows

⁵³ Benkler, *Wealth*, supra note 47, p.100

⁵⁴ *Ibid.*, p.101

⁵⁵ See Tim O’Reilly, “Lessons from Open-Source Software Development”, (1999) 42 (4) *Communications of the ACM* 33 at 37

⁵⁶ The “design” for FOSS projects is only for the purpose of aggregating peer-produced contributions together. It is not intended to micromanage contributors’ behaviour. In other words, the “design” should not undermine the flexible and improvisatorial nature of peer production.

⁵⁷ Richard Stallman, *GNU Emacs Manual* (Boston, MA: Free Software Foundation, 2010, 16th Edition)

⁵⁸ On top of this example of Linux, O’Reilly gives one example of the Perl programming language of extensibility: “Larry Wall “created Perl in such a way that its feature-set could evolve naturally, as human languages evolve, in response to the needs of its users.” Tim O’Reilly, “Lessons from Open-Source Software Development”, (1999) 42 (4) *Communications of the ACM* 33 at 37

Linux to be broken down into many “subsystems”.⁵⁹ All subsystems are stewarded by lead developers known as “maintainers” who have the responsibility to select, test and assemble contributions (known as “patches”) into a planned release. (A report published by the Linux Foundation in 2009 revealed that Linus Torvalds ranked number 9 among all subsystem maintainers in terms of his work of reviewing peer-produced code into the kernel.⁶⁰) Submitted patches must be reviewed and approved by the subsystem maintainers before they can be integrated into the Linux kernel. This means patches that fail to meet the expected standard may well be filtered out by the maintainers. Normally, maintainers need to give good reasons why some patches cannot be integrated into the project. The rejected programmers should be given the opportunity to appeal the decisions made against them. The rejection of patches would understandably cause much tension between the subsystem maintainers and the rejected programmers. So it is necessary to have “laws” and “courts” to efficiently solve disputes just like what is needed in the off-line real world.⁶¹ In the worst-case scenario, rejected programmers may “fork” or break away from the main project by starting up a new project in competition with the original one. In summary, the modular architecture in Linux does not just aggregate whatever contribution that is peer-produced, but it picks and chooses the most suitable ones that can be integrated into a coherent whole.

The second type of design decision that lead developers have to make for their collaborative project is a legal one: how to design a legal structure that makes a project “legally” extensible? The exiting copyright regime seems to be rather

⁵⁹ Weber observes that a modular architecture is “the key characteristic of technical design for managing complexity” for the Linux system: “Source code modularization obviously reduces the complexity of the system overall because it limits the reverberations that might spread out from a code change in a highly interdependent and tightly coupled system. Clearly it is a powerful way to facilitate working in parallel on many different parts of the software at once. In fact parallel distributed innovation is nearly dependent on this kind of design because a programmer needs to be able to experiment with a specific module of code without continually creating problems for (or having always to anticipate the innovations of) other programmers working on other modules.” Weber, *Success*, supra note 5, pp.172-5

⁶⁰ Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update* at <<http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf>>

⁶¹ The FOSS world is by no means anarchy, but it is governed almost as if it is an offline polity. Sanger observes: “In short, a collaborative community would do well to think of itself as a polity with everything that that entails: a representative legislative, a competent and fair judiciary, and an effective executive, all defined in advance by a charter. There are special requirements of nearly every serious community, however, best served by relevant experts; and so I think a prominent role for the relevant experts should be written into the charter.” Sanger, supra note 42, p.329

unfriendly to, if not militate emphatically against, collaboration on a radical level. Copyright permits authors to exclusively copy, modify and distribute the original software, and these permissions do *not* automatically extend to non-copyright holders. In an extensible FOSS project, a large number of participants are expected to frequently modify each other's works, and it is a hugely cumbersome job for each participant to clear the right each time a contribution is made however small it is. Even if people do tirelessly give some kind of permission when releasing their contributions, it is not guaranteed that all contributed modules are legally compatible with each other. For example, participant *A* allows his module *X* to be freely modifiable but not redistributable. Participant *B* allows his module *Y* to be both modifiable and re-distributable but at the same time he requires attribution to him in each of the downstream redistribution. So are *X* and *Y* *legally* compatible modules even though they can be *technically* welded into one piece of software? There is an even trickier situation: what if *A* or *B* changes his mind and decides to pull his contribution out of the project? Will he be allowed to go back on his commitment? Short of a formal and written agreement, these issues may well bog down the development of a collaborative project.

In order to avoid the above problems, standard form FOSS licences are designed to give some level of legal certainty. These licences are attempts to configure a legal structure (based on copyright law) to standardise the legal commitments of individual collaborators. They are designed to clear the legal hurdle for radical collaboration in FOSS projects. As has been shown in Section 1.2, a lead developer or a team of core developers need to choose between two types of licensing schemes, which are effectively two major legal designs for a given project: the copyleft design and the permissive BSD-style design. Copyleft is designed to make all downstream developers share their publicly released improvements of the original software in collaborative projects. It is an approach that suits well the community-based projects consisting mainly of voluntary contributors (such as GNU software and Linux projects), as it provides a guarantee that no contribution will be made proprietary in future distributions. In contrast, the permissive licensing design does not provide an anti-privatisation guarantee. Developers who choose this legal design are normally backed by well-funded established institutions including universities and software companies. Especially for those corporations writing software for consumer products

(e.g. operating systems for smartphones⁶²), they nowadays tend to license their products under a permissive licensing scheme. Their targeted users are normally non-sophisticated end-users who do not read or write code and they are not expected to make modification of the software. The non-copyleft design imposes virtually no restriction on redistribution of the software⁶³, which may help the software to spread quickly far and wide. For example, Google's release of the "Chrome" web browser in 2008 is a case in point. Chrome was licensed under the BSD License⁶⁴ clearly in an attempt to compete against the market incumbents like Microsoft's Internet Explorer and Mozilla's Firefox. This move is based on the speculation that the non-copyleft design can help Google's new software to break into the market rapidly and establish a big user-base in a relatively short space of time.

It is not difficult to see that between the two legal designs, copyleft dovetails better with the volunteer-driven peer-production model, while the permissive non-copyleft design nowadays tends to be adopted by for-profit corporations with a clear aim to expand their market. Both designs need a great deal of *coordination* in order to channel collaborative efforts into certain useful and coherent products. Coordination under copyleft tends to be led by volunteer developers who are rightly the members from the "peers". In contrast, the non-copyleft design relies less on peer-produced contributions, and its coordination may well be led by corporations that are keen to break their products into the market. This dissertation focuses itself mainly on the legal mechanism of the copyleft design (especially the GPL) for peer-produced FOSS projects, but the corporate strategic use of the non-copyleft scheme to build up a user base should not go unnoticed.

1.3.2 The Role of FOSS Licensing

A FOSS licence plays an important role in facilitating collaboration under the modular architecture of a corresponding project. It makes sure that peer produced contributions are legally compatible with each other and they can be safely pieced together into one integrated artifact. In other words, a FOSS licence is designed with

⁶² It is interesting to see that almost all mainstream smart phones (except Apple's iPhone) have adopted FOSS operating systems. Google's open-source Android system is a prominent example.

⁶³ However, it is also important to note many permissive licences still require retaining the copyright notice and correcting attribution in downstream distribution.

⁶⁴ <<http://www.google.com/chrome>>

an attempt to give some degree of legal certainty to collaboration in a FOSS project. When a collaborative project grows in size, the importance of a licence will accordingly increase, because the licence stabilises the programmers' expectation by spelling out what responsibility they should bear and what benefit they may also gain from the project. Furthermore, it is also worth noting that a licence by itself does not kick-start or constitute any collaborative relation. A licensing scheme is only the legal expression supporting the collaborative efforts that have already been going on or are expected to take place among all contributors.⁶⁵ The purpose of a FOSS licence is to verbalise and standardise the minimum legal commitments of a large number of existing and potential contributors. The expected legal commitments are rightly embodied in the form of FOSS stewardship responsibility pursuant to the Free Software Definition or Open Source Definition.

“Death of Assent” after the *ProCD* ruling

Almost all FOSS licences are non-negotiated standard form licences. They are mostly either in the form of clickwrap or browsewrap. The former requires users to manifest their assent by clicking button “Yes, I Agree” and the latter is just a webpage displaying the licensing terms and conditions. In terms of users' manifestation of assents, there seems to be no pronounced difference between FOSS and proprietary software that also uses mass-market off-the-shelf licensing schemes. In both cases, their respective standard form licensing schemes suffer from a similar problem, which is the lack of adequate assents from users.⁶⁶ In 1996, the US Seventh Circuit ruled that a shrink-wrap licence (as the precursor of clickwrap and browsewrap) was contractually enforceable in *ProCD v. Zeidenberg*⁶⁷, which then caused much heated debate.⁶⁸ This ruling signals what Lemley calls “death of assent”

⁶⁵ In fact, many collaborative activities may well precede the decision to formally adopting a licensing scheme.

⁶⁶ Clickwrap licences require higher level of manifested assent from users than browsewrap, but they may well still result in inadequate assents from users, who are unlikely to read and fully digest the content of the concerned licences before clicking through. Also, the FOSS world does not prefer clickwrap to browsewrap. The Criterion 10 of the Open Source Definition makes it clear that an open source licence cannot mandate the use of clickwrap in future redistribution of the software.

⁶⁷ 86 F.3d 1447 (7th Cir.1996)

⁶⁸ Scholars are divided on this case. For those who defend the court ruling, see, for example, Randy Barnett, “Consenting to Form Contracts” (2002) 71 *Fordham Law Review* 627 or Eric Posner, “*ProCD v Zeidenberg* and Cognitive Overload in Contractual Bargaining” (2010) 77 *The University of Chicago Law Review* 1181. For those who disagree with the ruling, see, for example, Stewart

in the digital environment where online standard form licensing becomes so ubiquitous:

Assent by both parties to the terms of a contract has long been the fundamental principle animating contract law. Indeed, it is the concept of assent that gives contracts legitimacy and distinguishes them from private legislation. But in today's electronic environment, the requirement of assent has withered away to the point where a majority of courts now reject any requirement that a party take any action at all demonstrating agreement to or even awareness of terms in order to be bound by those terms.⁶⁹

No doubt the death of assent in standard form poses a great challenge to the consent-driven classical contract law that enforces bilateral bargained exchanges. Neither is it good news to software users who are likely to be on the receiving end of this challenge. If assents cease to be the reason to legitimate non-negotiated obligations against licensees, then is there a new ground for enforcing these standard form FOSS licences? If there is one, what is it? The current dominant but not uncontroversial theory is based on a neoclassical economic justification that is articulated by the *ProCD* court. It goes that if a standard form licence makes an information product available to consumers at the lowest possible economic cost, then it should be justified and enforced.⁷⁰ However, the situation in a FOSS project is a bit more complicated than just a matter of maximising the economic utility for FOSS users but there is also considerable non-economically measurable social benefit that should not be ignored in a collaborative FOSS project. In Chapter 4, I will show that justifications other than material wealth maximisation (such as “software freedom” for its own sake) should also be considered in examining FOSS licensing terms in more detail.

Macaulay, “Freedom from Contract: Solutions in Search of a Problem?” (2004) *Wisconsin Law Review* 777 or Deborah W. Post, “Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook”, (2000) 16 *Touro Law Review* 1205. For Easterbrook's own explanation of the ruling, see Frank Easterbrook, “Contract and Copyright” (2005) 42 (4) *Houston Law Review* 953

⁶⁹ Mark Lemley, “Terms of Use” (2006) 91 *Minnesota Law Review* 459, at 464-5

⁷⁰ Whitford argues this rationale of wealth maximisation should not be the only yardstick against which the legitimacy of standard forms should be assessed. Other values (such as the norm of “participation” as proposed by Whitford) should also be considered under a relational contract perspective. See William C. Whitford, “Ian Macneil's Contribution to Contracts Scholarship”, (1985) *Wisconsin Law Review* 545 at 553-4

Collaboration and Relational Contract

In order to tackle the conundrum arising from the death of assent, I propose to examine FOSS licensing with an alternative approach—Relational Contract Theory (RCT)—which has remained conspicuously absent in the literature of FOSS licensing. RCT is a reaction to the classical view of contract as “abstract statements of the total obligation” leading to one-shot discrete transactions where the “parties may not have dealt before, and there is no assurance that they will deal again”⁷¹. In contrast, a relational contract is underpinned by pre-existing and ongoing relations where parties agree “to cooperate to achieve mutually desired goals.”⁷² My basic point here is that licences for collaborative FOSS projects cannot be discrete transactional contracts, but they need to be relational contracts that are supported by the pre-existing or ongoing collaborative relations experienced among participating contributors.

There are two important reasons why FOSS licensing should be closely scrutinised under RCT.⁷³ First, RCT posits that human beings have “dual motives” when engaged in a contractual exchange: they do not merely 1) seek to maximise their individual utility but 2) they also want to build “social solidarity” with other members of the society. In a relational contract, the second motive for social solidarity is especially important, because it reins in the otherwise unbridled first motive for utility maximisation.⁷⁴ Note that the first motive for utility is not eliminated altogether but it is only restricted by the second motive for solidarity. The double motives were well present in the collaborative ethos since the early computer hacker community. On the one hand, hackers enhanced their *utility* when each of them could have total and unlimited access to the continuous improvements of the software by other fellow-hackers. On the other hand, hackers bonded with each other through the practice of software sharing, which enhanced the *solidarity* within the community. However, the advent of proprietary software started a new trend where

⁷¹ Stewart Macaulay, “The Real and Paper Deal: Empirical Pictures of Relationships, Complexity and the Urge for Transparent Simple Rules” (2003) 66 *Modern Law Review* 44 at 65

⁷² *ibid.*

⁷³ In this introductory chapter, I only highlight these reasons for the relevance of RCT. This is intended to set the scene for a more detailed RCT analysis in Chapter 4.

⁷⁴ Ian R. Macneil, “Exchange Revisited: Individual Utility and Social Solidarity”, (1986) 96 (3) *Ethics* 567

the second motive for solidarity was gradually being swallowed into the first motive for utility. Proprietary software developers do not intend to establish collaborative relations with outsiders, but they are only interested in maximising their utility through selling as many as possible closed-source software products as if they were discrete commodities. Friends become strangers after a commodity transaction is consummated. Stallman observes that many programmers feel disheartened when their programming activities are reduced to a single motive for making money, while the motive for solidarity (or “friendship among programmers” in Stallman’s language) is jettisoned. In *The GNU Manifesto*, Stallman briefly analyses the consequence of loss of solidarity after the operating system software is close-sourced and commercialised:

Many programmers are unhappy about the commercialization of [operating] system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends.⁷⁵

The introduction of FOSS licensing is exactly an attempt to restore the balance between the dual motives for enhancing both utility and solidity, which is lost in commercial proprietary software. In this sense, FOSS licensing schemes are also an effort to rebuild the relational contract among programmers under the software stewardship tradition. Furthermore, I need to caution my readers that the dual motives in a relational contract are not necessarily mutually exclusive. I do not wish, by quoting the above paragraph from Stallman, to give a misimpression: one has to sacrifice one motive for the other. In fact, the two motives in FOSS collaboration are often closely connected and mutually reinforcing. For example, the reputational gains play a hugely important role in incentivising production of FOSS in a collaborative manner. Satisfaction from one’s enhanced reputation as a kind of non-monetary reward is an interesting and somewhat ambivalent motive. On the one hand, the reputation of one’s virtuosity in programming and generosity in sharing

⁷⁵ Stallman, *The GNU Manifesto*, 1985, at <<http://www.gnu.org/gnu/manifesto.html>>

contributions results from the recognition from other community members, and it is essentially a product of enhanced solidarity. On the other hand, this good reputation may also increase one's material utility in the real world. For example, it is likely to increase a programmer's employability to get a permanent salaried job.⁷⁶ In this sense, the reputational reward for writing FOSS is the site where the boundary between the two motives is blurred in a relational contract.

The second reason for RCT's relevance to FOSS licensing lies in its position on the role of consent (or assent) in contractual exchange. In a classical contract, obligation is presumed to arise in a single moment where there is a meeting of the minds between parties. Macneil finds this consent-centred view is not helpful in leading to a more realistic understanding of contractual exchanges: "The dominant role of consent in the jurisprudence of classical contract law has put intellectual barriers in the way of communicating a broader analysis of the subject that appears in that jurisprudence."⁷⁷ In contrast, RCT has a more nuanced position on this issue. A relational contract is less driven by explicit consents (or assents to standard form licence in particular), but obligation may also come out of parties' experience from the pre-existing and ongoing relations.⁷⁸ So in a FOSS project, the pre-existing and ongoing collaborative relations become highly important in the sense that they will alleviate the burden on discrete explicit consents as the sole device to effectuate the obligations in a corresponding licence. In other words, it should not be ignored that relations can also give rise to obligations when explicit consents are weak or non-existent in standard form licences.

It is also important to be aware that assents in FOSS licensing are not irreversibly "dead" but they are just being relationalised. (Gudel similarly observes that there is no "decline of assent," but there is only "a decline of assent *discretely* understood" in

⁷⁶ It is not rare that many lead FOSS programmers are later hired by software companies in recognition of their exceptional programming talent and leadership quality. See Eric Raymond, *Homesteading the Noosphere*, supra note 26

⁷⁷ Macneil, *The New Social Contract—An Inquiry into Modern Contractual Relations* (New Haven and London: Yale University Press, 1980), pp.47-48 (Hereafter *NSC*)

⁷⁸ It is observed that Macneil's message that "there is no single moment at which the parties confirm a meeting of the minds respecting the important terms of the contract" has been relatively well accepted by contract scholarship. William Whitford, "Ian Macneil's Contribution to Contracts Scholarship", supra note 69, at 546

a more general context.⁷⁹) At its worst, consents, as Macneil claims, still function as “a vital triggering mechanism” in contractual exchanges.⁸⁰ As there are a growing number of FOSS projects that are nowadays targeted at non-programming end-users rather than sophisticated co-developers, manifested assents through clickwrap licences do not become entirely unnecessary. Because these end-users do not directly participate in the collaborative relation of co-developing a certain FOSS product, they should be given a good chance to know what kind of licensing scheme they will enter into and there is no harm in doing so.

Collaboration and Intellectual Property

The relational contract perspective offers important insights into the role of collaborative relations in FOSS licensing. However, FOSS collaborative relations do not merely come out of the pristine software stewardship tradition originated from the hacker custom, but they have also been deeply affected (both positively and negatively) by the institution of intellectual property in software since the 1980s. (Recall that FOSS licensing is a compromise between these two conflicting traditions of stewardship and private property.) Barnett argues that “property” is also “a highly relational concept that performs its own vital social functions”⁸¹, but Macneil’s RCT has never adequately developed a line of theoretical inquiry into the role of “property” in relational contract:

I maintain that although [Macneil’s] observation [“standing behind all relational exchange or contracts is a socially-enforced system of property socially-enforced system of property”] is largely true, somewhat surprisingly [property] is never properly integrated into Macneil’s social analysis. Consequently his social theory of contract is virtually, if not entirely, uninfluenced by any comparable social theory of property. Related to this is the near complete

⁷⁹ Paul J. Gudel, “Relational Contract Theory and the Concept of Exchange”, (1998) 46 *Buffalo Law Review* 763 at 773

⁸⁰ Macneil, *NSC*, supra note 77, p.50

⁸¹ Randy Barnett, “Conflicting Visions: A Critique of Ian Macneil’s Relational Theory of Contract”, (1992) 78 (5) *Virginia Law Review* 1175 at 1181

absence in his theory of background rights that can be used to evaluate normatively the legal rights actually recognized by a legal system.⁸²

Contractual exchanges are not made in vacuum, but they are profoundly shaped by “background rights” as delineated by “property”, which is allegedly neglected by Macneil. In other words, property precedes and continues through contractual exchanges. Even if these contractual relations come to an end, the property relation will keep living on. Benkler’s definition of “property” serves as a good example of the relational aspect of “property” that provides the background rules for defining relations between property owners and the non-owning public:

Property is a cluster of *background rules* that determine what resources each of us has when we come into *relations* with others, and, no less important ‘having’ or ‘lack’ a resource entails in our *relations* with these others. These rules impose constraints on who can do what in the domain of actions that require access to resources that are the subject of property law.⁸³ (added emphasis)

Note that property posits an *asymmetrical* relation between owners and the non-owning public in Benkler’s definition. The asymmetry is due to the exclusive rights given to property owners, who are then entitled to exercise unilateral power over non-owners in terms of utilising the owned resources. Benkler makes it clear that property rules “are aimed to crystallize *asymmetries* of power over resources, which then form the basis for exchanges.”⁸⁴ (added emphasis)

Compared with the asymmetrical “property” relation, “commons” is intended to be a symmetrical arrangement. The symmetry in the relation under a commons has twofold meanings. First, all participating members have equal non-exclusive rights to the resources within a particular commons. Secondly, all participants have the same obligations to other members of the commons. The purpose of FOSS licensing is exactly to re-configure the asymmetries posited by intellectual property with an attempt to create a symmetrical relation among all members of software commons. In this software commons, all software developers have the same set of rights (or

⁸² *ibid.*, 1180-1

⁸³ Yochai Benkler, *Wealth*, supra note 47, p.143

⁸⁴ *ibid.*

software freedoms) to access, use, modify and redistribute software. At the same time, all of them are bound by the same set of duties, i.e. the stewardship responsibility, to refrain from exercising some of their exclusive rights given by intellectual property. In a nutshell, FOSS licensing is based upon the institution of “property”, but it reconfigures the asymmetric property relation into a symmetrical one under a software commons where property owners and the non-owning public share the same set of rights and duties.

The radically decentralised collaboration of any FOSS project is impossible without the symmetrical relation under a software commons, where everyone can legally make and share improvements of the original software. This symmetrical arrangement under software commons has two perceived advantages. On the one hand, individual programmers enhance their utility by being able to use a rapidly improved software program. On the other hand, they also enhance the solidarity with other members of the commons through sharing contributions to a FOSS project. These two advantages are not readily available from the asymmetrical relation within a proprietary software project.

Finally, it is worth knowing that total and despotic ownership does *not* actually exist in software copyright. Copyright does make some effort to keep a level of symmetry between software authors and their users. There are numerous occasions where software can be utilised without the copyright holders’ permission.⁸⁵ It is a balance that is needed to rein in the unilateral power that may be exercised by copyright holders to unfairly restrict non-owning public’s rights. For example, under the UK copyright law, “reverse engineering” or “decompilation” to achieve interoperability between programs is a permitted act by any lawful software user.⁸⁶ So in this particular respect, copyright creates a symmetrical relation between copyright holders and lawful users. However, it is not rare for a proprietary software licence to include a clause forbidding reverse engineering or decompilation for any purpose.⁸⁷

⁸⁵ Lemley’s points out that copyright makes “a number of compromises between the desires of authors and those of the consuming public”. Mark Lemley, “Beyond Preemption: The Law and Policy of Intellectual Property Licensing” (1999) 87 (1) *California Law Review* 111 at 128

⁸⁶ In the UK, there is a statutory right for a lawful user to “decompile the program to obtain the information necessary to create an independent program which can be operated with the program decompiled or with another program”. This right cannot be contracted out. s.50 B, CDPA 1988

⁸⁷ Lemley, “Beyond Preemption”, supra note 85 at 128

By doing so, they destroy the symmetrical relation by shrinking the commons and expanding their private ownership interests in software beyond copyright. Benkler calls this kind of behaviour “contractual enclosure” of the software commons.⁸⁸ In contrast, FOSS licensing schemes go down the opposite direction of “contractual enclosure” by enlarging software commons through shrinking software copyright holders’ exclusive rights.

1.4 Structure of the Dissertation

This opening chapter has set the scene for the exploration of the three aspects of FOSS licensing that is used in support of radically decentralised FOSS collaboration. The rest of the dissertation will continue this exploration of the historical aspect (in Chapter 2), legal aspect (in Chapters 3 and 4) and authorial aspect (in Chapter 5) of FOSS licensing and its role in collaboration in more detail and depth.

Chapter 2 traces the historical development of FOSS licensing. It identifies three historical stages during which the early computer Hacker Ethic begun, evolved, and matured into software stewardship obligations detailed by FOSS licences. Chapter 3 examines how FOSS programmers struggle to articulate a legal expression of software freedom through the device of software licensing. It focuses on two areas of “intellectual property”—copyright and patent—and their respective threat to software freedom. It uses the GPL as an example to show how FOSS programmers assess possible threats to software freedom respectively from copyright and patent and how they attempt to contain these threats through many generations of the GPL since its inception until the latest 2007 version. Chapter 4 tackles some difficult issues concerning the FOSS licences as non-negotiated standard form contracts from a Relational Contract Theory (RCT) perspective. It attacks the neoclassical contract approach (as represented in the *ProCD* ruling) that has been dominant in the mainstream software licensing jurisprudence. It tries to demonstrate that RCT is a more suitable theoretical tool to analyse FOSS licensing schemes as a legal means to support relation-rich FOSS projects. Chapter 5 examines FOSS authorship at both individual and collective (project) levels. It shows how FOSS programmers manifest their authorial consciousness through their licensing scheme. The focus will be on

⁸⁸ Benkler, *Wealth*, supra note 47, p.444

project leaders' legal persona as author-stewards for their collaborative projects. Chapter 6 summarises three aspects of FOSS licensing in relation to this dissertation's contribution to the scholarly literature and it also points out two avenues to future research.

Chapter 2 From the Hacker Ethic to “Open Source”: A Brief History

2.1 Introduction: Three Historical Stages

FOSS licences that are used by programmers in collaboration do not appear suddenly in a historical vacuum. The idea of radically decentralised collaboration was fermented at the very beginning of computer hacker culture, and it took decades for its unique Hacker Ethic to evolve into today’s FOSS licences in their fully-fledged form. The evolution from the Hacker Ethic to FOSS licensing schemes is by no means a smooth succession of discrete events, but it is complicated and contentious. In order to do full justice to the complexity of the topic, this chapter sketches out three historical stages during which the early computer Hacker Ethic began, evolved, and matured into software stewardship obligations detailed by FOSS licences. The first stage starting in the 1950s till the early 1980s is the pre-licensing period when collaboration among programmers was based on the Hacker Ethic. This Hacker Ethic was challenged by the rise of proprietary software and then the early hacker community underwent gradual disintegration when many computer hackers were hired away to write proprietary software. The second stage spanning a period from early 1980s to 1998 witnessed the birth and growth of a most influential copyleft licence—GNU General Public Licence (GPL)—which tried to translate some elements of the lost Hacker Ethic into a legally binding document. The ingenuity of the GPL lies in its copyleft mechanism which is an anti-privatisation device to ensure publicly distributed code to always remain in software commons. The third stage started with 1998 when the movement of “open source” was officially launched by Eric Raymond and his colleagues, who intended to integrate non-proprietary software into the commercial mainstream. This period witnesses the growing commercial and legal strength of open source that can compete with proprietary software products.

It is also an important task of this chapter to show the subtly different characteristics of collaborative relations in building FOSS projects at the three historical stages. Very briefly, in the first stage, collaboration was largely forged by the non-binding

Hacker Ethic that took place in a relatively organic and spontaneous fashion from the old hacker community. In the second stage, the old collaborative efforts were largely disrupted by the rise of proprietary software. The disruption prompted free software programmers to craft their own licensing schemes in order to repair the damaged community-based collaborative relations. In the third stage, the prospect of making money out of “open source” software attracted an increasing number of corporate collaborators to join various projects. Although it was impossible to restore the hacker custom to its original purity, FOSS licences would at least play a role in preventing for-profit companies from entirely dictating or “recentralising” the production and circulation of FOSS in what was meant to be a radically decentralised collaborative environment.

There are two caveats about historicising the development of FOSS collaboration and licensing in this chapter. First, my account of the three historical stages cannot be a chronicle of every single factual event, but concentrates only on the conceptual trajectory along which stewardship and private ownership in software have co-evolved to have an impact on FOSS licensing. Secondly, the three stages are not necessarily discretely separated from each other but they can also be seen as a continuum where one stage shades into the next. For example, Linux is exactly a cross-stage project, which had its pre-life as the pedagogical Minix system derived from the UNIX operating system in the first stage, and it took off as a viable GPLed product in the second stage, and then was showcased as a continuously successful “open source” product in the third stage. Again, there is no natural clear-cut demarcation line in history, but I do wish to highlight some of the most critical events (such as the Emacs dispute that prompted Stallman to write the GPL) in order to bookmark the changes that are critical in the development of FOSS licensing. To appreciate three stages as a continuous whole would lead to a rounded understanding of FOSS licensing as a legal phenomenon in its historical context. This understanding will form the foundation for the analysis of the legal mechanism of FOSS licensing that leverages intellectual property law to preserve software commons in the following chapters.

2.2 From the 1950s to the Early 1980s: The Pre-Licensing Era

The first historical stage can be roughly subdivided into two halves. The first half witnesses the formation and growth of the Hacker Ethic from the 1950s to the mid-1970s. This ethic was a moral code stipulating hackers' duty to share information for the sake of solving technical problems collaboratively. The second half of this stage covers a period from the mid-1970s to the early 1980s, when the Hacker Ethic was challenged and eclipsed by an emerging new norm of "owning" software as private intellectual property. Collaboration in this stage does not depend on any licensing scheme that could restrict privatisation of software, but it only resorted to the non-legally binding moral force of the Hacker Ethic, which was becoming nonetheless increasingly vulnerable to the encroachment of proprietary software.

2.2.1 Beginning of the Hacker Ethic

The software stewardship tradition began in the computer hacker community which was mostly based in a few US academic institutions such as the Massachusetts Institute of Technology (MIT) in the 1950s and 1960s. This tradition is embodied in the Hacker Ethic, which was dutifully observed and carried out in full measure by programmers well into the early 1970s. Richard Stallman recalls that when he first joined the MIT Artificial Intelligence (AI) Lab in 1971, he naturally "became part of a software-sharing community that had existed for many years." The software-sharing ethic, according to him, is "as old as computers, just as sharing of recipes is as old as cooking."¹ Though the norm of software-sharing was ubiquitous during that period, the term "free software" did not exist and there was no need for one. This is because intellectual property law such as copyright had not yet been extended to software and there was no need to differentiate "free" from "proprietary" software. Stallman explains:

We did not call our software 'free software,' because that term did not exist, but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using

¹ Stallman, "The GNU Operating System and the Free Software Movement" in *Open Sources: Voices from the Open Source Revolution* eds. by Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O'Reilly & Associates, 1999) p.53

an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.²

The Hacker Ethic of software sharing that Stallman witnessed and experienced in the AI Lab since 1971 is important in two senses. First, it provides a shared body of rules that define who the hackers are and what they should do. Second, it forms the ethical foundation of the stewardship obligations that will later make their way into Stallman’s copyleft licensing scheme. However, for a long time, the Hacker Ethic remained largely unwritten and it is said to be “an ethic seldom codified, but embodied instead in the behaviour of hackers themselves.”³ The difficulty of studying this ethic exactly lies in the difficulty of pinning down a rather fluid body of unwritten norms which are only known by hackers themselves and are much less visible and obvious to outsiders. (The later FOSS licences mitigate this problem by writing down what exactly are the core sets of obligations that hackers should bear.) With the benefit of hindsight, Steven Levy’s 1984 book *Hackers—Heroes of the Computer Revolution* (hereafter *Hackers*) was the first attempt to systemically document the Hack Ethic that was originally formulated in the 1950s and the 1960s. Levy identifies six tenets of the Hacker Ethic and they are organised around the first tenet known as the “Hands-on Imperative”, which encourages hackers to share information by allowing “unlimited and total” access to computers. The six tenets are:

- Access to computers—and anything which might teach you something about the way the world works—should be unlimited and total. Always yield to the Hands-on Imperative!
- All information should be free.
- Mistrust Authority—Promote Decentralisation.
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position.
- You can create art and beauty on a computer.
- Computers can change your life for the better.

The first tenet, which is often shortened to “Hands-on Imperative”, is a *sine qua non* for computer hackers to solve engineering problems and then share solutions in a most effective and collaborative way. It is based on the fact that hackers are first and

² *ibid.*

³ Levy, *Hackers*, p.7

foremost “engineers” who make computer machines work⁴, and computer hacking (as well as the later “open source” programming) starts exactly in “an engineering culture” that is “grounded heavily in experience rather than theory.”⁵ Hackers’ “thinking” is not merely conducted through pure theoretical speculation, but it is more closely derived from their engineers’ instinct to fix or tweak defective machines. The sociologist Richard Sennett, based on his observation of Linux developers, argues that FOSS programmers are not unlike traditional “craftsmen” who engage in practical manual work: they are craftsman-like technicians who conduct “a dialogue between concrete practice and thinking” and “this dialogue evolves into sustaining habits, and these habits establish a rhythm between problem solving and problem finding.”⁶ Suppose that the “unlimited and total access” to computers was obstructed, this dialogue between “concrete practice and thinking” would be severely disrupted. Levy also explains the importance of this first tenet that comes out of programmer-engineers’ practical need to experiment with things including computers: “Hackers believe that essential lessons can be learned about the systems—about the world—from taking things apart, seeing how they work, and using this knowledge to create new and even more interesting things. They resent any person, physical barrier, or law that tries to keep them from doing this.”⁷

The rest of the five tenets are essentially under the umbrella of the first tenet. The second tenet that mandates an unobstructed free flow of information (“all information should be free”) is clearly a corollary of the Hands-on Imperative. Levy’s commentary on this tenet is in the form of a rhetorical question: “If you do not have access to the information you need to improve things, how can you fix them?” The answer is that “[a] free exchange of information, particularly when the information was in the form of a computer program, allowed for greater overall creativity.”⁸ Weber observes that Stallman later became one of the most ardent

⁴ Copyright analogises programmers to literary writers, because they write human-readable source code. This preoccupation sometimes obscures the fact that computer hackers are also primarily problem-solving engineers whose code can be executed by machines. see Pamela Samuelson, Randall Davis, Mitchell D. Kapor, J. H. Reichman, *A Manifesto Concerning the Legal Protection of Computer Programs* (1994) 94 (8) *Columbia Law Review* 2308 (Hereafter *Manifesto*)

⁵ Steven Weber, *The Success of Open Source* (Cambridge, Mass.: Harvard Uni. Press, 2004) p.164 (Hereafter *Success*)

⁶ Richard Sennett, *The Craftsman* (New Haven & London: Yale University Press, 2008) p.9

⁷ Levy, *Hackers*, p.40

⁸ *ibid.*

supporters of this tenet, which would have huge consequence on the free software movement.⁹ However, it is also important to know that the later development of FOSS licensing shows that the informational freedom is *not* an absolute freedom, but it can be circumscribed in an environment affected by intellectual property (IP). Wagner argues information freedom in “open source” is achieved through the controlled use of IP: “the ‘open’ in open source is actually rather tightly controlled, albeit in the name of generally greater access along certain philosophically favored dimension. And it is fundamentally the control of intellectual property rights that allows such arrangements to be struck.”¹⁰ Furthermore, Raymond also warns that not all information should necessarily be free, especially that which is related to individuals’ privacy.¹¹ In this light, my thesis is built upon a nuanced understanding of the second tenet, which means that all information should be free to the extent that programmers can freely collaborate to build a common project.

The third tenet registers hackers’ great dislike of centralised authority and their advocacy for decentralisation. It is squarely targeted at centralised bureaucratic systems, including corporations, government and universities, because they are believed to be “dangerous” and “cannot accommodate the exploratory impulse of true hackers.”¹² (Ironically, in the 1960s, IBM was seen by hackers as an epitome of this danger of centralisation,¹³ though it later turned out to be an important corporate participant in the open source movement in the third historical stage.) This anti-centralisation tenet also anticipates the radically decentralised Bazaar-type open-source production as opposed to the centralised Cathedral-type software

⁹ Weber’s commentary of Tenet 2 specifically mentions Stallman’s role in promoting the second tenet: “Richard Stallman would later become the most vocal champion of the principle that software, as an information tool that is used to create new things of value, should flow as freely through social systems as data flows through a microprocessor.” Weber, *Success*, supra note 5, p.144

¹⁰ Note what Wagner discusses here is a slight variation of the second tenet: “information wants to be free” See Polk Wagner, “Information Wants to Be Free—Intellectual Property and the Mythologies of Control” (2003) 102 *Columbia Law Review* 995 in *Intellectual Property: Critical Concepts in Law*, edited by David Vaver (Oxford: Routledge, 2006) p.351

¹¹ Raymond points out that “[s]ome kinds of information really do want to be free, in the weak sense that their value goes up as more people have access to them—a technical standards document is a good example. but the myth that all[] information wants to be free is readily exploded by considering the value of information that constitutes a privileged pointer to a rivalrous good—a treasure map, say, or a Swiss bank account number, or a claim on services such as a computer account password. Even though the claiming information can be duplicated at zero cost, the item being claimed cannot be. Hence, the non-zero marginal cost for the item can be inherited by the claiming information.” Eric Raymond, *Magic Cauldron*, at <<http://www.catb.org/~esr/writings/magic-cauldron/>>

¹² Levy, *Hackers*, p.41

¹³ *ibid*, pp.41-3

manufacturing, which is a distinction drawn by Raymond many years later.¹⁴ It also has its reincarnations in later academic discussions such as those about “peer production”¹⁵ or “Wikinomics”¹⁶ in terms of the radically decentralised way of creating information enabled by networked computer technology.

The fourth tenet envisions that the hackerdom should be strictly built upon a meritocracy where hackers “should be judged by their hacking not bogus criteria such as degrees, age, race, or position.” It is made clear that conventional non-hacking related credentials are superficial and irrelevant, and that what hackers can contribute to the community matters the most. “This meritocratic trait was not necessarily rooted in the inherent goodness of hacker hearts—it was mainly that hackers cared less about someone’s superficial characteristics than they did about his potential to advance the general state of hacking, to create new programs to admire, to talk about that new feature in the system.”¹⁷ This tenet shows hackers’ longing for their hackerdom to be an autonomous sphere independent from the “real” non-hacking world. It also tallies with Raymond’s observation that the most able and devoted hackers tend to get more reputational reward than others in a collaborative project.¹⁸

The fifth tenet concerns the aesthetic of programming. It says that hacking is not just a mindless technical job but it can also involve “art and beauty on a computer”. Recall that in the first tenet, hackers are first and foremost craftsmen or technicians. However, there can be a very thin line between craftsmanship and art. Hackers can move beyond coding as craftsmanship and they become programming artists by writing code ‘elegantly’. The aesthetic dimension of coding makes programmers

¹⁴ Raymond, *The Cathedral and the Bazaar* (hereafter *Cathedral*) version 3.0 at <<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>>

¹⁵ Benkler, *Wealth of Networks: How Social Production Transforms Markets and Freedom*, (New Haven: Yale University Press, 2006)

¹⁶ Don Tapscott and Anthony D. Williams, *Wikinomics* (London: Portfolio, 2006)

¹⁷ Levy, *Hackers*, p.43

¹⁸ Eric Raymond, “Homesteading the Noosphere”, 2002 at <<http://www.catb.org/~esr/writings/homesteading/homesteading/>>

appear rather like literary authors, who may imprint their creative personality into their works.¹⁹

The sixth tenet believes that computers “can change your life for the better.” It sends an evangelical message that computer technologies would not only benefit computer hackers but also more broadly the whole of humanity. The Hacker Ethic should spread outside “the monastic confines of the Massachusetts Institutes of Technology” and reach the non-programming part of the society. “If *everyone* could interact with computers with the same innocent, productive, creative impulse that hackers did, the Hacker Ethic might spread through society like a benevolent ripple, and computers would indeed change the world for the better.”²⁰ (original emphasis) This tenet is corroborated by the later development of “free culture”²¹ and “cultural environmentalism”²², where the Hacker Ethic of information sharing spills over into non-programming creative spheres enabled by networked computer technology. For example, projects such as Wikipedia are among the most successful applications of this tenet beyond software.

In summary, the Hacker Ethic identified by Levy portrays a picture of what computer programmers in the 1950s and the 1960s thought about themselves. The six tenets form the “shared identity and belief system” that would underpin hackers’ core set of common commitments to building software projects collaboratively.²³ Though they were not legally binding but only voluntarily observed by computer hackers themselves at this stage, they started the hacker stewardship tradition which would form the ethical foundation for the later FOSS licensing schemes.

¹⁹ For the discussion of the analogy of computer programmers and literary authors, see Clapes, Lynch, and Steinberg, “Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs” (1987) 34 *UCLA Law Review* 1493

²⁰ Levy, *Hackers*, p.49

²¹ Lessig, Lawrence, *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity* (New York: The Penguin Press, 2004)

²² James Boyle, “Cultural Environmentalism and Beyond” (2007) 70 *Law and Contemporary Problems* 5; Molly Shaffer Houweling, “Cultural Environmentalism and the Constructed Commons”, (2007) 70 *Law and Contemporary Problems* 23

²³ The Hacker Ethic as a belief system is also thought to be one of the many motivational factors that drive FOSS programmers to write software. The six tenets of the Hacker Ethic, according to Weber, would have huge impact on the later “open source” movement as they “continue to characterize the open source community to a surprising degree.” Weber, *Success*, supra note 5, p.144

2.2.2 Decline of the Hacker Ethic

In the mid-1970s the Hacker Ethic began to be eroded by a new norm of owning software as private property. Software programmers started to feel proprietorial about software that was written and many of them stopped sharing code with other programmers. In 1976, the then young Bill Gates²⁴, in the capacity of General Partner of Microsoft, authored an open letter, accusing other computer hackers of being property-stealing hobbyists. Gates's letter can be boiled down to an argument that writing software was not a matter of indulging one's curiosity, but it involves professional programmers' hard labour that should be economically rewarded. It openly challenged the Hacker Ethic of information sharing: "As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?"²⁵ In short, the production of software for Gates is a serious business that requires economic incentives. The letter was, during his time, considered to be tactless,²⁶ and it was littered with blunt accusatory words such as "stealing" and "theft":

One thing [hobbyists] do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little *incentive* to make this software available to hobbyists. Most directly, the thing you do is theft.²⁷ (added emphasis)

It is important to learn that this open letter was not produced *ex nihilo*, but it was an outlet of his anger after a specific incident that Gates encountered. Before the letter was published, Microsoft, under the partnership of Gates and Paul Allen, produced a

²⁴ Gates was also a featured computer hacker in Steven Levy's study of the hacking community in the 1970s. Gates was described as "Cocky wizard, Harvard dropout who wrote Altair BASIC, and complained when hackers copied it." Steven Levy, *Hackers*, p.10

²⁵ Bill Gates, "An Open Letter to Hobbyists", 3 February 1976 at <http://www.digibarn.com/collections/newsletters/homebrew/V2_01/gatesletter.html>

²⁶ The letter was written and published without first consulting Ed Roberts who actually employed Gates to write the Altair BASIC. See Levy, *Hackers*, pp.229-230

²⁷ Bill Gates, "An Open Letter to Hobbyists", supra note 24

popular version of the BASIC computer language. It ran on the microcomputer known as Altair, which was a precursor to mass-manufactured personal computers (PC) for less sophisticated end-users. Countering the Hacker Ethic, Gates insisted on a new norm that the Microsoft version of Altair BASIC should be paid for instead of being shared and copied free of charge. This suggestion did not go down well with the hackers who were still in their old sharing habit at that time. For example, those hackers who were members of the Homebrew Computer Club were among the most enthusiastic sharers of Altair BASIC.²⁸ Gates felt extremely frustrated when very few people actually sent payment to him after using the software:

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however, 1) Most of these ‘users’ never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent on Altair BASIC worth less than \$2 an hour.²⁹

Given the later extraordinary commercial success of Microsoft’s proprietary software, Gates’ letter is often retrospectively singled out as a notable bookmark signaling the future sea-change of the old Hacker Ethic giving way to the norm of proprietary software.³⁰ This letter is significant also in the sense that for the first time a new norm against the Hacker Ethic was emphatically verbalised in a widely circulated written document.³¹ To be more precise, the significance has twofold meaning. First,

²⁸ Homebrew Computer Club, founded in the mid-1970s, was active in sharing software among its members. The club incubated important hackers such as Steve Wozniak, who later made huge contribution to the development of affordable domestic microcomputers. The club members were enthusiastic in “sharing” Altair BASIC and they were exactly the kind of “hobbyists” that Gates’ 1976 open letter criticised. Levy observes: “People around the Homebrew Computer Club tried to ease into this new era, in which software had commercial value, without losing the hacker ideal. One way to do that was by writing programs with the specific idea of distributing them in the informal, though quasi-legal, manner by which Altair BASIC was distributed—through a branching, give-it-to-your-friends scheme. So software could continue being an organic process, with the original author launching the program code on a journey that would see an endless round of improvements”. Levy, *Hackers*, pp.230-1

²⁹ Gates, *supra* note 24

³⁰ For example, Steven Levy devotes half a chapter to discussing Gates’ letter, which is used as a piece of written evidence showing the “the new fragility of the Hacker Ethic”. Levy, *Hackers*, pp.224-237; Steven Weber also quotes the letter as the “alternative tracks” to the Hacker Ethic in the mid-1970s, see Weber, *Success*, *supra* note 5, pp.35-37

³¹ David Bunnell, the then editor of *Altair Users’ Newsletter*, managed to circulate the letter in many places including Homebrew Computer Club’s newsletter. Levy, *Hackers*, p.229

Gates envisioned that software could be neatly separated from hardware and be sold on its own as commodity. This is different from the world of the old Hacker Ethic where there was “no meaningful distinction between hardware and software” and “code was the machine”.³² (Recall that “computers” in the Hands-on Imperative calling for the unlimited and total access refers to both hardware and software programs.) In order to pave the way for full commodification of software, Gates’ letter challenges the norm that “Hardware must be paid for, but software is something to share”. He wished to elevate software to the status of being fully alienable commodity in its own right. Secondly, the open letter’s repeated use of words like “steal” and “theft” indicates that Microsoft BASIC started to be interpreted as a kind of private property exclusively belonging to its authors. Gates here clearly was advocating a new norm of private ownership in software, which was radically new and disturbing in 1976. Boyle sees Gates’s open letter as an attempt to drive home a basic point that “software needs to be protected by (enforceable) property rights if we expect it to be effectively and sustainably produced”³³. This new norm is at least four years ahead of its time because US Congress would not amend its copyright legislation to cover software until 1980. The letter also raised the issue of economic “incentive” for producing software, which was closely related to the orthodox understanding of the function of private property as the reward of authors’ labour. Following Gates’s logic, short of a system that could exclude members of the public from copying Microsoft BASIC, the incentive for programmers to write this software cannot be really guaranteed. In short, the open letter contains the seminal idea that software should become fully fledged private property, which would pave the way for the full commodification of software in the future.

Gates’ open letter no doubt dropped a bombshell on the hackerdom, but it would be an exaggeration to say that it directly led to the demise of the MIT-style hacker

³² “As in the early days of computing, the code was the machine in a real sense. And code was something you naturally collaborated on and shared. This was natural because everyone was just trying to get their boxes to do new and interesting things, reasonably quickly, and without reinventing the wheel.” Weber, *Success*, supra note 5, p.36

³³ James Boyle, *The Public Domain—Enclosing the Commons of the Mind* (New Haven& London: Yale University Press, 2008) p.164

community.³⁴ Hackers' reaction to Gates' letter was extremely negative. Only five or six people were persuaded to send Gates the payment that was insisted in the open letter.³⁵ The Hacker Ethic was not immediately defeated but it would take another half of a decade for its fragility to be fully exposed.³⁶ However, it would be safe to say that after 1976 there started to emerge two camps of software programmers. One includes the old "ethical" MIT-style computer hackers who shared everything with their fellow-hackers and the other attracts the more business-minded programmers who wanted to sell software to make profit. It also creates a schism where two competing norms that would eventually run into intense clash. The former claims that it is programmers' stewardship duty to share software and the latter insists that there are private property subsisting in software from which programmers should be economically rewarded. This is a tension suggesting that neither stewardship nor private property is natural to software. Software as "property" has always been a hotly contested social construct and it cannot be impervious to the changing social milieu. Weber observes the camp sticking to the old Hacker Ethic and the camp following the new norm of software ownership would battle for supremacy endlessly from then on: "Both sides claimed (and continue to claim) that their worldview was self-evident, obvious, and an inevitable consequence of the material forces and constraints that exist in computing. But neither proprietary nor free software is 'blind destiny.' Both continue to coexist, in a kind of software industry 'dualism' [...]. Neither is a technological necessity, and neither can claim to have 'won out' in any meaningful sense."³⁷

³⁴ The MIT hacking environment collapsed largely due to a company called Symbolics, which hired away many of its hacker. I will discuss the incident in more detail in sub-Section 2.3.2

³⁵ Levy quotes Hal Singer as a representative voice against Gate's letter: "the most logical action was to tear the letter up and forget about it." Levy, *Hackers*, p.230

³⁶ Levy observes that the Hacker Ethic still lingered on after the 1976 open letter: "When MIT hackers were writing software and leaving it in the drawer for others to work on, they did not have the temptation of royalties. [...] With the growing number of computers in use (not only Altairs but others as well), a good piece of software became something which could make a lot of money—if hackers did not consider it well within their province to pirate the software. No one seemed to object to a software author getting something for his work—but neither did the hacker want to let go of the idea that computer programs belonged to everybody. It was too much a part of the hacker dream to abandon." *ibid.*

³⁷ Steven Weber, *Success*, supra note 5, p.37

2.3 From the Early 1980s to 1998: Clash between the Two Traditions

The second stage started in the 1980s when the Hacker Ethic was further weakened by the rising norm of proprietary software. The schism between the two camps of old MIT-style hackers and Microsoft-type proprietary programmers was further enlarged by the new legislation of US copyright law to include software. The ascendancy of private ownership in software effortlessly eclipsed the old software stewardship tradition. In response to this change, Richard Stallman almost singlehandedly started the “free software” campaign in order to regain some lost ground of the Hacker Ethic. Apart from continuing to writing non-proprietary software, Stallman also crafted the very first “copyleft” software licence as an anti-privatisation device to ensure that modified versions of free software cannot be subjected to a proprietary regime. Copyleft must be used in conformity with Free Software Definition, whose spirit is derived largely from the first two tenets of the Hacker Ethic.

In the following subsection, I will show two contexts in which a selected few important changes have affected hackers’ collaborative relations in the second historical stage. The first is a broad context in which the new norm of private ownership in software gradually gained ascendancy. It concerns the changing economic situation and legal environment that became more conducive to commoditisation of software since the 1980s. The second is the narrow and specific context situated in the MIT AI Lab where Richard Stallman as an individual reacted to the trend of privatisation of software by devising a copyleft licence in order to repair the broken collaborative relations among hackers. I will explain both contexts in turn.

2.3.1 Changes in Market and Law

The broad context contains two elements, of which one is about market and the other about law. The new market situation, combined with the new legal environment, helped software to metamorphose into a kind of exclusive and fully alienable “property” in its most orthodox sense.³⁸ Firstly, there emerged a reasonably big

³⁸ Conventionally, a thing becomes private property if it has two attributes: 1) “alienability” and 2) “exclusivity”. To put it crudely, if a thing is allowed to be bought and sold, it is alienable; if it can exclude others from accessing or using it, it is exclusively owned. See Roger Smith, *Property Law*

market for software products (especially operating systems for microcomputers) to be traded as commodities in large quantities. This was the kind of market that did not exist in the first historical stage where software was shared within the hacker community. Most significantly, thanks to the plummeting cost of hardware components, the advent of affordable personal computers (PC) for domestic use substantially increased the number of computer users with varied levels of computing literacy. Many of these new users possessed little or no programming knowledge at all and they did not have to. For these non-sophisticated users, what they need was *workable* software more than *modifiable* software and they were properly *end-users*. This is a crucial shift because in the older hacking community there was no clear distinction between programmers and users, i.e., everyone is actually or potentially a co-developer. In the new situation, a new group of end-users was created and they only *consume* software. Proprietary software thrives on the “dumbing down” of computer culture, because it creates a market where there are consumers more willing to pay for readymade software but less curious and inquisitive about the technology beneath it.

Secondly, the legal environment in the 1980s also changed dramatically in favour of programmers who wanted to exert exclusive control of their software over users. This was the beginning of an era when a body of law known as “intellectual property” was developed to take software under protection. Most significantly, software, across the Atlantic, was made eligible for copyright protection. In 1980, the US Congress amended the 1976 Copyright Act to include software as a subject matter.³⁹ In the UK, the 1985 Copyright (Computer Software) Amendment analogised software to the literary works protected by the 1956 Copyright Act. Three years later, software unequivocally became a protectable subject matter as a species of “literary work” in under the Copyright, Designs and Patents Act 1988.⁴⁰

The introduction of software copyright appeared upsetting to Richard Stallman who was steeped in the Hacker Ethic. Though Stallman later changed his view about

(Essex, England: Pearson Education Limited, 2003, 4th Edition) p.3 Thanks to the economic and legal changes in the 1980s, it was possible for commercial proprietary software acquired both attributes.

³⁹ 17 U.S.C. s.101

⁴⁰ Section 3(1), CDPA 1988

copyright, his then knee-jerk reaction was viscerally negative. When he first encountered programs displaying “copyright notices” on the screen, he thought that they were “blasphemy”⁴¹ to the Hacker Ethic. Stallman was against treating software programs as literary works because he thought they were fundamentally different. In 1985, he pointed out that software (containing both human-readable source code and executable object code) and literary works (i.e. books) were different in the sense that the former could instruct computer to perform certain functions and the latter were merely literary text to be read. There was little harm to copyright literary works as such, but to copyright software programs would lead to the result of “harming society as a whole”:

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of non-fiction. This practice was useful, and is the only way many authors’ works have survived even in part. The copyright was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books. [...] The case of programs is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object which are distinct, and the fact that the *a program is used rather read and enjoyed*, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the [copyright] law enables him to.⁴² (added emphasis)

Stallman’s argument that copyright is not an ideal legal form that should regulate software on the ground that “a program is used rather than read and enjoyed” is not entirely unfamiliar to legal scholars, some of whom suggest replacing copyright with *sui generis* software protection for very similar reasons.⁴³ We will soon find that

⁴¹ Levy, *Hackers*, p.419

⁴² Stallman “The GNU Manifesto”, 1985, at <<http://www.gnu.org/gnu/manifesto.html>>

⁴³ For example, Samuelson and her co-authors hold a similar view that programs are not merely literary text but more importantly they are instructions for machines. Pamela Samuelson *et. al.*, *Manifesto*, supra note 4

Stallman later became less hostile to copyright, after he discovered that the broad exclusive rights granted to software authors could actually be leveraged to deter privatisation of software in a “copyleft” licensing arrangement.⁴⁴

Another legal development is to protect software under trade secrecy, which became an increasingly common practice after the 1970s⁴⁵. This development affects two groups of people: the first group includes programmers and the second includes users. First, in order to get software under trade secrecy protection, proprietary software companies needed to make sure that their own employees do not leak and spread the source code outside. So programmers-employees were asked to sign non-disclosure agreement on the software they developed. Second, proprietary developers no longer release software with the source code available to users. The executable code-only software usually came with a proprietary software licence forbidding reverse engineering altogether. However, non-sophisticated end-users tended to accept this change, because they did not really care much about whether the source code was kept secret or not. Recall that widespread PCs had a “dumbing down” effect in the computer world where a lot of PC users did not read, let alone modify, source code of software. Feller and Fitzgerald find that the move of distributing only non-human readable object code brought a convenient result to both non-programming end-users and commercial software developers. For end-users, when millions-of-line source code is compiled into object code, it saved a lot of storage space, which was still very precious on 1980s’ microcomputers; for software developers, source code, when kept secret, became a valuable asset in its own right, and it effectively prevent competitors from knowing how their software was actually coded.⁴⁶

Looking back, Gates’ norm, that payment must be made for software for its own sake, sounded radical and unfamiliar in 1976, but it became commonplace in the 1980s. The new market situation and legal environment combine to create an atmosphere conducive to the production of more profitable proprietary software. Software was effectively unbundled from hardware, and it can be traded in its own right. Copyright

⁴⁴ See for detail in sub-Section 2.3.2

⁴⁵ Mark Lemley, “Convergence in the Law of Software Copyright”, (1995) 10 *High Technology Law Journal* 1 at 4

⁴⁶ Joseph Feller and Brian Fitzgerald, *Understanding Open Source Software Development*, (London: Addison-Wesley, 2002) pp.11-12

and trade secrecy gave software a legal basis to exclude non-paying users. Moglen observes that the changes clearly gave rise to the “right to exclude” that was desired by proprietary software developers.

After 1980, everything was different. The world of mainframe hardware gave way within ten years to the world of the commodity PC. And, as a contingency of the industry’s development, the single most important element of the software running on that commodity PC, the operating system, became the sole significant product of a company that made no hardware. High-quality basic software ceased to be part of the product-differentiation strategy of hardware manufacturers. Instead, a firm with an overwhelming share of the market, and with the near-monopolist’s ordinary absence of interest in fostering diversity, set the practices for the software industry. In such a context, *the right to exclude others from participation in the product’s formation became profoundly important*. Microsoft’s power in the market rested entirely on its ownership of the Windows source code.⁴⁷ (added emphasis)

The changes in market and law give the broad context of the rise of exclusive private property in software and the decline of the Hacker Ethic. It explains how software programmers gradually became the exclusive owners of the software that was produced. However, this broad context does not explain much how the very first “copyleft” free software licence known as the GNU General Public Licence was produced and how it was employed to save the declining Hacker Ethic. I now need to move to the more specific context based at the MIT AI Lab where Richard Stallman would react by inventing copyleft as an anti-privatisation device to rebuild collaborative relations among programmers.

2.3.2 The Birth of Copyleft

It is difficult to identify one single moment when Stallman conceived the idea of copyleft. However, I endeavour to highlight, with the benefit of hindsight, three crucial incidents that precipitated the invention of copyleft. The significance of these incidents might not have been fully apparent at the time when they took place, but

⁴⁷ Eben Moglen, “Anarchism Triumphant: Free software and the Death of Copyright”, (1999) 4 (8) *First Monday* at <http://www.firstmonday.org/issues/issue4_8/moglen/>

retrospectively they were so frequently told that they became an indelible part of free software developers' collective memory. The three stories share a common theme that runs through a narrative explaining the birth of copyleft: Copyleft is a hacker's reaction to the rise of proprietary software and it is an attempt to rebuild the collaborative ethos under the Hacker Ethic by means of software licensing.

The Xerox Printer Incident

The first incident is a much repeated story involving a Xerox paper-jammed printer.⁴⁸ In around 1980, Stallman at the MIT AI Lab was using a cutting edge laser printer donated by Xerox.⁴⁹ He encountered a glitch which failed to allow him to print out a 50-page file. With the hacker's typical "Hands-on Imperative", Stallman felt compelled to identify and fix the problem immediately. "As a person who spent the bulk of his days and nights improving the efficiency of machines and the software programs that controlled them, Stallman felt *a natural urge* to open up the machine, look at the guts, and seek out the root of the problem."⁵⁰ Stallman's "natural urge" is emblematic of the first tenet of the Hacker Ethic identified by Levy: "Always yield to the Hands-on Imperative", which spurs a hacker "to fix something that [...] is broken or needs improvement."⁵¹ Unfortunately, Stallman was not able to track down the source-code file because Xerox this time did not provide it to the AI Lab as was the case before. Stallman vividly recalls his frustration when he could not access and modify the source code many years later:

Later Xerox gave the AI Lab a newer, faster printer, one of the first laser printers. It was driven by proprietary software that ran in a separate dedicated computer, so we couldn't add any of our favorite features. We could arrange to

⁴⁸ Stallman repeated the story in many places and he used this personal experience as an example of deterioration of the Hacker Ethic that he could feel at the MIT AI Lab. Sam Williams wrote the first book-length biography of Stallman. William dedicated the whole opening chapter to only to the Xerox printer incident in great detail and shows its significance on the Stallman's cause. See Williams, Chapter 1, *Stallman's Crusade, Free as in Freedom--Richard Stallman's Crusade for Free Software*, O'Reilly 2002 at <<http://www.oreilly.com/openbook/freedom/>> (hereafter *Stallman's Crusade*)

⁴⁹ The older Xerox graphics printer was initially donated to the AI Lab in around 1977. The printer "was run by free software to which we added many convenient features. For example, the software would notify a user immediately on completion of a print job. Whenever the printer had trouble, such as a paper jam or running out of paper, the software would immediately notify all users who had print jobs queued. These features facilitated smooth operation." Stallman, "Why Software Should be Free" at <<http://www.gnu.org/philosophy/shouldbefree.html>>

⁵⁰ Sam Williams, *Stallman's Crusade*, supra note 47, para. 29

⁵¹ See Levy, *Hackers*, p.40

send a notification when a print job was sent to the dedicated computer, but not when the job was actually printed (and the delay was usually considerable). There was no way to find out when the job was actually printed; you could only guess. And no one was informed when there was a paper jam, so the printer often went for an hour without being fixed.

The system programmers at the AI Lab were capable of fixing such problems, probably as capable as the original authors of the program. Xerox was uninterested in fixing them, and chose to prevent us, so we were forced to accept the problems. They were never fixed.⁵²

Stallman later learned that a leading computer scientist just left Xerox and was hired by Carnegie Mellon University's computer science department. He made a journey to Carnegie Mellon and made a request in person for the source-code file that ran the printer. Much to Stallman's disappointment, the request was turned down to his face. This is because the ex-Xerox employee had already signed a non-disclosure agreement with Xerox and the source code must be kept as the trade secret of the company.⁵³ Stallman felt emotionally scarred by the refusal to provide source code by another programmer. From this experience, he finds that unmodifiable proprietary software would cause a "psychosocial harm" to software users just like a resident is not allowed to make any changes to a house where lives: "It is demoralizing to live in a house that you cannot rearrange to suit your needs. It leads to resignation and discouragement, which can spread to affect other aspects of one's life. People who feel this way are unhappy and do not do good work."⁵⁴

The Symbolics Incident

Although the Xerox incident has been repeatedly singled out as "a major turning point" when proprietary software started to hurt the collaborative relations under the Hacker Ethic, it is nothing more than a wake-up call about the creeping influence of

⁵² Stallman, "Why Software Should be Free" at <<http://www.gnu.org/philosophy/shouldbefree.html>>

⁵³ It is suspected that Robert Sproull was the one who rejected Stallman's request of source code, but Sproull himself had no recollection of the incident. Williams, *Stallman's Crusade*, supra note 47, para. 58.

⁵⁴ supra note 52

proprietary software on the MIT Lab.⁵⁵ The real “fatal blow” that destroyed the Stallman’s hacker community came from the second incident about “Symbolics”, which was a spin-off company from the MIT AI Lab. In the early 1980s, Russell Noftsker, a former AI Lab administrator, formed Symbolics to commercialise an AI Lab project on the LISP programming language. Its business competitor was the hacker-friendly company called LISP Machine Incorporation (LMI) led by the MIT hacker Richard Greenblatt, who stuck to the Hacker Ethic and disclosed their source code dutifully as usual. In a nutshell, there were three parties in this incident: Symbolics, LMI and the AI Lab. The first two parties were in business competition and the last was in a neutral position. Stallman’s job at the AI Lab was to keep the lab’s version of the LISP operating system abreast with the two companies’ improvements. The three parties shared improvements of LISP OS for over a year. In March 1982, Symbolics stopped sharing source code with the AI Lab and LMI in order to protect their software as trade secret. This was a move that was intended to undermine its competitor LMI, but the person who felt most betrayed and hurt was Stallman at the AI Lab. Stallman’s personal revenge was to reverse engineer Symbolics’ now “closed-source” software by studying their newly added features, whose source code would then be completely rewritten from scratch by Stallman.⁵⁶ He then shared his code with Symbolics’ competitor the LMI. Angered by Stallman’s retaliation, the President of Symbolics Noftsker accused Stallman of “stealing” the company’s trade secrets:

We developed a program or an advancement to our operating system and make it work, and that may take three months, and then under our agreement with MIT, we give that to them. And then [Stallman] compares it with the old ones and looks at that and see how it works and reimplements [for the LMI

⁵⁵ Williams observes that the incident at its most is a “wake-up call” alarming that software “had become such a valuable asset that companies no longer felt the need to publicize source code” Sam Williams, *supra* note 47, para. 62 & para. 65,

⁵⁶ Retrospectively, Stallman’s method of simulating functions of the original software by rewriting source code was later called “non-literal copying” of software. Stallman thought his behaviour was perfectly legal because he did not literally copy source code. However, non-literal copying was held to be infringing a software owner’s copyright in 1986 *Whelan Associates Inc. v. Jaslow Dental Laboratory Inc.* 797 F.2d 1222 (3d Cir. 1986); The *Whelan* decision was criticised by a later decision for giving an overbroad protection of software, *Computer Associates Int’l, Inc. v. Altai, Inc.*, 928 F.2d 693 (2d Cir.1992)

machines]. He calls it reverse engineering. *We call it theft of trade secrets.*⁵⁷
(added emphasis)

Stallman's revenge did not go very far. Many fellow-hackers at the MIT Lab had already disagreed with him and they were gradually hired away to write more lucrative proprietary software for Symbolics.⁵⁸ They saw Stallman's reaction as a "troubling anachronism" which was blind to the irreversible trend of software commercialisation: "In commercializing the Lisp Machine, the company pushed hacker principles of engineer-driven software design out of the ivory-tower confines of the AI Lab and into the corporate marketplace where manager-driven design principles held sway."⁵⁹ The upshot of the Symbolics incident is the full decline of the Hacker Ethic at the AI Lab. For Stallman, the hacker-led AI Lab in the early 1970s was not only his workplace, but also his spiritual home. Stallman held Symbolics responsible for destroying this "home". Just as Williams observes that "the Symbolics controversy dredged up a new kind of anger, the anger of a person about to lose his home."⁶⁰ As a "homeless" hacker, Stallman felt that there was no point in continuing to work at the AI Lab where the hackers' collaborative ethos no longer existed. He resigned his job and became a full-time campaigner for "software freedom".

The Emacs Programming Editor Dispute

The third and final incident concerns Stallman's dispute with James Gosling over the GNU Emacs software, which eventually led to the creation of the first copyleft licence in the period between 1983 and 1985. After Stallman left the AI Lab, he embarked on the ambitious GNU project in order to create a complete non-proprietary operating system to replace the proprietary UNIX system. In an initial announcement dated 27 September 1983, Stallman planned to "write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it

⁵⁷ Quoted by Levy, *Hackers*, p.426-7

⁵⁸ Even before the Symbolics incidents, Stallman was already shunned by many hackers who later joined Symbolics. In around 1981 and 1982, MIT hackers stopped inviting Stallman to go out for dinners together and some of them confessed that they had to lie to Stallman in order to avoid embarrassment. Sam Williams, *supra* note 47, para. 455

⁵⁹ *ibid.*, para. 454

⁶⁰ Williams observes that Stallman felt strongly that MIT AI Lab in the early 1970s was his "home": Williams, *ibid.*, para. 456

away free to everyone who can use it.”⁶¹ In the same announcement, Stallman stressed a “golden rule” of software sharing that he himself must stick to: “I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement.” More significantly, in the following paragraph, the word “free software” was mentioned: “So that I can continue to use computers without violating my principles, I have decided to put together a sufficient body of *free software* so that I will be able to get along without any software that is not free.”⁶² (added emphasis) Although it would take a few more years for “free software” to be clearly defined (as in the FSD), the announcement shows Stallman was contemplating the idea of “free software” as early as in 1983.

One of the flagship sub-projects of GNU is the Emacs programming editor, which was initially developed by Stallman (in collaboration with many of his colleagues at the AI Lab) in the mid-1970s. Emacs embodied a then radically new idea of displaying and editing text on computer screens (replacing the old method of scrutinising printed-out code on paper). It was a pioneer of “the real-time display editor” and it was “customizable” by its users. The customizability of Emacs was the most outstanding feature intended by Stallman to facilitate radical collaboration between developers and users. According to Levy, Stallman “used the Hacker Ethic as a guiding principle for his best-known work, an editing program called EMACS which allowed users to limitlessly customize it—its wide open architecture encouraged people to add to it, improve it endlessly.”⁶³ Very importantly, users are given a tool known as the Emacs Lisp (Elisp) programming language to make any adaptation that they need. The official website of GNU Emacs explains:

If Emacs doesn't work the way you'd like, you can use the Emacs Lisp (Elisp) language to customize Emacs, automate common tasks, or add new features. Elisp is very easy to get started with and yet remarkably powerful: you can use

⁶¹ The project was slightly delayed, and it did not make a start until January 1984. Stallman, *GNU Initial Announcement*, 1983 at <<http://www.gnu.org/gnu/initial-announcement.html>>

⁶² Stallman, *GNU Initial Announcement*, 1983 at <<http://www.gnu.org/gnu/initial-announcement.html>>

⁶³ Levy, *Hackers*, p.416

it to alter and extend almost any feature of Emacs. You can make Emacs whatever you want it to be by writing Emacs code [...] ⁶⁴

Because of its radical openness and customizability, Emacs became so popular among programmers that it was widely copied and modified in various forms. Stallman suggested that anyone who made improvements of the Emacs editor should contribute modifications back to the so-called “EMACS software-sharing commune”. In a 1981 Emacs user manual, it became clear that Stallman had by then conceived a prototypical “share-alike” condition for using Emacs, though it was not intended to be legally binding:

[...] you are joining the EMACS software-sharing commune. The conditions of membership of [the EMACS commune] are that you must send back any improvements you make to EMACS, including any libraries you write, and that you must not redistribute the system except exactly as you got it, complete. [...] All sources [i.e. source code] are distributed, and should be on line at every site so that users can read them and copy code from them. [...] ⁶⁵

The Emacs commune “share-alike” condition ⁶⁶ stipulating that “you must send back any improvements you make to EMACS” is significant. The Emacs commune, according to Kelty, is “designed to keep EMACS alive and growing as well as to provide it for free” and it indicates a kind of “community stewardship” ⁶⁷, which is different from private ownership of software. It fleshed out the first two tenets of the Hacker Ethic documented by Levy. ⁶⁸ It would take another four years before Stallman actually wrote a fully-fledged copyleft licence—GNU Emacs General Licence—in 1985, but the idea of “copyleft” was clearly being fermented as early as

⁶⁴ *A Guided Tour of Emacs*, <<http://www.gnu.org/software/emacs/tour/>>

⁶⁵ Stallman, “EMACS Manual for ITS Users,” 22 October 1981, quoted in Kelty, *Two Bits*, Note 13, p.333

⁶⁶ Elsewhere, Stallman addresses this Emacs commune condition in a less formal way: the condition is “that [users] give back all extension they made, so as to help EMACS to improve. I called this arrangement ‘the EMACS commune’”. Stallman continues that “[a]s I shared, it was their [i.e. users’] duty to share; to work with each other rather than against.” quoted in Levy, *Hackers*, p.416

⁶⁷ Kelty, *Two Bits*, p.191

⁶⁸ Recall that the first two tenets are the “hands-on Imperative” (i.e., unlimited and total access to computers) and “all information should be free”. Stallman’s verbalisation of the commune conditions in 1981 gives the Hacker Ethic a more concrete textual existence.

in 1981 when copyright was about to become an established legal form of software protection.

Unfortunately, not everyone shared Stallman's ideal of "community stewardship" of the Emacs editor. Since 1981, James Gosling started to work on a variant of Emacs running under the UNIX operating system. Gosling's version (sometimes also known as "Gosmacs") was initially shared with the community and Stallman incorporated some of Gosling's code into his GNU Emacs. In 1983, Gosling decided not to share his software any more, on the ground that the increasing popularity of Gosmacs made him unable to keep up with the growing administrative side of the job.⁶⁹ He eventually sold Gosmacs to a proprietary software company called Unipress, which was believed to be more suitable for the future development of Gosmacs. Stallman was saddened by Gosling's decision and felt that his communal ideal was seriously eroded.

Furthermore, also around this period of time copyright gradually overtook trade secrecy as the main form of legal protection of software,⁷⁰ and the threat of copyright infringement became increasingly real rather than merely hypothetical to software developers. In order to avoid being bogged down by the copyright ownership dispute with Gosling, Stallman did two things. First, he removed Gosling's code completely from GNU Emacs and issued a "Gosling-free" version with his own replacement. Second, he produced the first free software licence that specified the conditions of using, modifying and redistributing GNU Emacs. The new licence was called GNU Emacs General Public License (EGPL), which was first published in 1985 and revised twice respectively in 1987 and in 1988. The licence opened with a preamble marking the difference between GNU Emacs and software produced by proprietary "software companies" (with Unipress clearly being one of them in Stallman's mind): "The license agreements of most software companies keep you at the mercy of those companies. By contrast, our general public license is intended to give everyone the right to share GNU Emacs. To make sure that you get the rights we want you to have,

⁶⁹ Gosling made this announcement on 12 April 1983: "This is a hard step to take, but I feel that it is necessary. I can no longer look after [Gosmacs] properly, there are too many demands on my time. EMACS has grown to be completely unmanageable. Its popularity has made it impossible to distribute free: just the task of writing tapes and stuffing them into envelopes is more than I can handle." quoted in Kelty, *Two Bits*, p.190

⁷⁰ See Kelty, *Two Bits*, pp.199-206

we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights.”⁷¹ Note that the “share-alike” clause of the EGPL is not qualitatively different from the content of the Emacs commune condition mentioned above. The EGPL’s “copying policies” stipulate that Emacs users must

cause the whole of any work that you distribute or publish, that in whole or in part contains or is a derivative of GNU Emacs or any part thereof, to be licensed at no charge to all third parties on terms identical to those contained in this License Agreement (except that you may choose to grant more extensive warranty protection to some or all third parties, at your option).⁷²

This clause would be later famously (or notoriously) known as the “viral” clause, which constitutes the defining feature of copyleft. It enjoined GNU Emacs users to contribute back any publicly released modification, i.e. any work that “in whole or in part contains or is a derivative of GNU Emacs”, under the same GNU EGPL. This was clearly designed to prevent programmers like Gosling from withdrawing their contributions from the Emacs commune.

Most significantly, in the course of the Emacs dispute from 1983 to 1985, Stallman’s attitude towards copyright underwent an important but sometimes unnoticeable change: he became less cynical about copyright in software and found that he could leverage copyright law to further his cause of free software.⁷³ It is important to remember that this happened in a historical context where an increasing number of software developers began to rely on copyright to protect their software.⁷⁴ The dispute with Gosling caused Stallman to gradually familiarise himself with the US copyright law. He discovered that the very broad right granted to copyright owners could actually be inflected for the “share-alike” purpose intended by the Emacs “commune”. He used copyright as the basis for imposing the “copyleft”⁷⁵

⁷¹ GNU EGPL

⁷² Section 2 (b), GNU EGPL, 1985, 1987, 1988 at <<http://www.cogsci.indiana.edu/pub/COPYING>>

⁷³ From 1983 when Gosling made Gosmacs proprietary to 1985 when Stallman published the first copyleft licence, the change took place in a short space of less than two years.

⁷⁴ For example, Kelty observes that both Gosling and Stallman registered their respective versions of Emacs with the US Library of Congress after the dispute. Kelty, *Two Bits*, Note 43, p.335

⁷⁵ In 1985, Stallman received a letter from Don Hopkins who wrote playfully on the envelope the phrase: “Copyleft—all right reversed”. Stallman liked the clever wordplay “copyleft” and later used it to name the anti-privatisation feature of the GNU EGPL-like software licences, which are known as

requirement—which is effectively a variant of the Emacs commune condition—on downstream users and deterred them from privatising source code of released modifications and improvements. In the same year when the 1985 GNU Emacs licence was published, Stallman also founded the Free Software Foundation (FSF), which later becomes an important powerbase for the free software movement.

In 1989, Stallman finally turned the Emacs-specific GNU EGPL into a generic template licence, which could be used for any free software. It became the very first version of GNU General Public Licence (GPL). Stallman made it clear that the GPL v1.0 relied upon copyright: “We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.”⁷⁶ This text would remain unchanged in all the later versions of the GPL. Legally speaking, the GPL is a kind of standard form licence that lacks explicit bargained-for exchanges. It is important to note that, despite its venerable ideal of protecting software freedom, the legal form of GPL is not drastically dissimilar from other conventional off-the-shelf standard form software licences, which are also used by proprietary software developers. In Chapter 4, I will try to tackle this issue in more depth by examining the doctrinal rules governing the enforcement of FOSS licences and I will also propose to analyse the issue by harnessing the insights from relational contract theory, which has been largely neglected in the literature of FOSS licensing.

To summarise the whole second historical period of FOSS licensing, the three incidents of the Xerox printer at the AI Lab, Stallman’s confrontation with Symbolics and the Emacs dispute travelled down a trajectory where Stallman formulated a narrative about the birth of the free software movement as an attempt to repair the collaborative ethos damaged by the rise of proprietary software. The narrative is closely linked with the development of intellectual property law in the US from the mid-1970s onward: the first two incidents are mainly concerned with protecting software by trade secrecy (through non-disclosure agreements), while the last one shifts to copyright protection. This shift signals a trend where copyright

the copyleft licences. Richard Stallman, “The GNU Project”, at <http://www.gnu.org/gnu/thegnuproject.html>

⁷⁶ Preamble, GPL v1.0; the quoted sentence remains unchanged in the second and third versions of GPL.

became a more convenient and accepted form of software protection than trade secrecy after 1980. It is not a surprise that Stallman's many versions of GPL licences are actually *copyright* licences because they are essentially the product of this trend. The next historical stage will reveal how the free software movement took another critical turn after 1998 and its social and political influence started to spill over into the non-hacking world as anticipated by the sixth tent of the Hacker Ethic.

2.4 From 1998 Onwards: Challenge from “Open Source”

The year 1998 marks the beginning of the third stage in the history of FOSS collaboration. In early February of that year, Eric Raymond, an ex-Emacs contributor, openly broke away from Stallman's free software movement. Raymond's agenda is to redirect the energy of free software developers to a more business-friendly approach under a new label called “open source”. The term was deliberately coined by Raymond on 3 February 1998 during a meeting with some entrepreneurially minded programmers in California. It was hoped that this “rebranding” of the movement as “open source” would tone down the anti-commercialism associated with the Stallman-led free software movement and get “open source” software accepted in the business world.⁷⁷

Deviating from Stallman's position, Raymond's neologism is not intended to hold the moral high ground over proprietary software, but it is keen to propagate a vision that the decentralised bazaar-style “open source” is capable of producing better-quality software than the hierarchically organised Cathedral-like structure. This vision was actually first gestated, one year before the coinage of the term, in the much-cited essay *The Cathedral and the Bazaar* first written in 1997 and later revised numerous times by Raymond. This essay is significant in the sense that it opened up an alternative line of narrative, deviating from Stallman's narrative of “software freedom” as a matter of regaining the lost ground of the MIT-style Hacker Ethic. The new narrative contains a series of carefully chosen stories, which would later become part of open source's own “folklore”. There are two oft-told stories that

⁷⁷ On that day, Raymond met Todd Anderson, Chris Peterson, John “maddog” Hall and Larry Augustin, Sam Ockman, Michael Tiemann in Palo Alto. The meeting was held after Netscape's announcement (in January) of its plan to release its source code to the public. Open Source Initiative, “History of the OSI”, at <<http://www.opensource.org/history>>

tellingly illustrate how Raymond takes trouble to introduce the new twist of “open source” to the “free software” world. The first is about the Linux kernel system as an epitome of the Bazaar model and the second is about the Netscape web browser as an example of corporations’ embrace of the idea of “open source”.

First, Raymond capitalises on the runaway success of the Linux kernel project, which already took off in the early 1990s.⁷⁸ One of the main goals that *The Cathedral and the Bazaar* wants to achieve is to further catapult Linus Torvalds, the initiator of Linux, to the centre-stage of the “open source” movement. Raymond opens his essay with a verdict that “Linux is subversive”, followed immediately by a thought-provoking question: “Who would have thought [in 1991] that a world-class operating system could coalesce as if by magic out of part-time hacking by several thousand developers scattered all over the planet, connected only by the tenuous strands of the Internet?”⁷⁹ Raymond admitted that he himself, before the mid-1996, failed to appreciate the strength of decentralised production of software until the success of Linux awakened him to the tremendous advantage brought by the open-source bazaar. He explains why the “subversive” Linux is crucial to the understanding of the “bazaar” model as opposed to the hierarchical “cathedral” structure.

Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized a priori approach was required. I believed that the most important software (operating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in a splendid isolation, with no beta to be released before its time.⁸⁰

Note that the above paragraph obliquely criticises programmers like Stallman for behaving like “individual wizards or small bands of mages working in splendid

⁷⁸ The Linux project took off well before *the Cathedral and the Bazaar* was written. So Linux inspires, but is not inspired by, the Bazaar model that Raymond attempt to advocate.

⁷⁹ Raymond, *Cathedral*, supra note 14

⁸⁰ *ibid.*

isolation”⁸¹ and it attacks Stallman’s Emacs as an example of “really large tools” that “needed to be built like cathedrals”.⁸² In contrast, Linus Torvalds does not work in “splendid isolation,” but his style of development—“release early and often, delegate everything you can, be open to the point of promiscuity”—resembles “a great babbling bazaar of differing agendas and approaches”.⁸³ Far from working alone, Torvalds, since he was a college student in Helsinki, did not hesitate to enlist the help of thousands of volunteer programmers to contribute to the Linux project. Raymond wants to emphasise that the success of Linux is a result of mass collaboration on the global scale. It does not matter much how talented each of the individual programmers is, but it does matter a lot how collaborative all contributors are.

It is worth noting that the way that programmers are portrayed in Raymond’s writing is markedly different from Levy’s journalism on hackers. Levy’s 1984 book *Hackers* is a collection of larger-than-life programming geniuses and its subtitle revealingly hailed them as the “Heroes of Computer Revolution”. In contrast, the younger generation of Linux programmers including Torvalds do not enjoy the “heroic” status that their predecessors have, but they are ordinary people who are just willing to work collaboratively “to the point of promiscuity” as is recorded in *The Cathedral and the Bazaar*. Again, a brief comparison of Levy’s portrait of Stallman and Raymond’s writing about Torvalds is illustrative of the more “democratic” and less elitist characteristic of the new “open source” movement. In Levy’s book, Stallman is portrayed as “the last of true hackers” whose heroism includes almost singlehandedly fighting for the lost Hacker Ethic. Recall that when Stallman joined the MIT AI Lab in 1971, he was still an undergraduate studying at an elitist Ivy League university.⁸⁴ The then AI Lab was likened by Levy to a “monastery”, where

⁸¹ Torvalds also voices his dislike of GNU Emacs and he openly says that “the Emacs editor is horrible”. Also, Torvalds’ adoption of GNU software tools for the Linux project is purely for practical reason and it is not a matter of buying into Stallman’s “philosophy”. See Torvalds, “The Linux Edge”, in *Open Sources: Voices from the Open Source Revolution* (O’Reilly & Associates, 1999) p.107

⁸² It is not the first time that Stallman was found to be an extremely capable but lonely hacker who often works in the solitarily environment. Bill Gosper, was aware of, and also admired, Stallman’s ability to work efficiently on his own. Stallman single-handedly rewrote the new features of the LISP operating system during his personal battle with the software company Symbolics. Gosper comments: “I can see something Stallman wrote, and I might decide it was bad (probably not, but someone could convince me it was bad), and I would still say, ‘But wait a minute—Stallman doesn’t have anybody to argue with all night over there. He’s working alone! It’s *incredible* anyone could do this alone.” quoted in Levy, *Hackers*, p.426

⁸³ Raymond, *Cathedral*, supra note 14

⁸⁴ He was earning a magna cum laude degree in physics at Harvard at the same time.

Stallman “had experienced the epiphany” of “pure hacker paradise” and thus developed “a deep affinity for the Hacker Ethic, and was militant in his execution of its principle.”⁸⁵ In contrast to the rarefied monastic atmosphere at MIT, Torvalds is said to write Linux in a more “promiscuous” and democratic environment, which is more conducive to mass collaboration from almost all walks of life. According to Raymond, most Emacs tools (with a couple of exceptions) are built by elitist cathedral-builders like Stallman, while Linux is a promiscuously open bazaar where individual programmers like Torvalds are “lazy as a fox.”⁸⁶ In particular, Raymond points out that Torvalds is not an exceptionally original programmer. The Linux system is heavily derivative from the pre-existing Minix system and there involves no “conceptual leap forward” from Minix to Linux as something radically novel. Compared with the older-generation hackers like Stallman, Torvalds is by no means an “innovative genius” of programming (and he does not have to be one), but his main contribution lies in his ability to select, implement and reuse and piece together other people’s code. Though not a programming genius, Torvalds is recognised as a kind of lesser “genius” who is exceptionally good at more mundane tasks of “engineering and implementation” of other contributors’ ideas. This is a recognition of the importance of Torvalds’s role as coordinator of a large-scale radically decentralised collaborative project:

[...] Linux didn’t represent any awesome conceptual leap forward. Linus [Torvalds] is not (at least, not yet) an innovative genius of design in the way that, say, Richard Stallman or James Gosling [...] are. Rather, Linus seems to me to be a genius of engineering and implementation, with a sixth sense for avoiding bugs and development dead-ends and a true knack for finding the minimum-effort path from point A to point B. indeed, the whole design of Linux breathes this quality and mirrors Linus’s essentially conservative and simplifying design approach.⁸⁷

Although Raymond’s comparisons between the Cathedral and the Bazaar, between Linux and Emacs, and between Torvalds and Stallman, are not universally accepted

⁸⁵ Levy, *Hackers*, pp.415-6

⁸⁶ Raymond, *The Cathedral*, supra note 14

⁸⁷ *ibid.*

among critics⁸⁸, Raymond largely succeeded in achieving what he wanted to achieve: to promote an easy-to-grasp image of what “open source” is for the general public (especially the non-programming business world) by showing how the decentralised Bazaar model works and how it might be applied to collaborative projects like Linux.

The second important story in Raymond’s narrative of “open source” is about Netscape, a software company that was keen to apply the Linux model. It is an example of a high-profile corporation abandoning the Cathedral model for the new Bazaar model. On 23 January 1998, the Netscape management team took a brave decision to release the source code of their flagship product—the web browser known as Navigator—to the public and became an “open source” company.⁸⁹ This move was taken in reaction to a dire prospect that was faced by Netscape when their browser was rapidly losing market share to Microsoft, which bundled its Internet Explorer browser to its Windows operating system. Before decision was made, Frank Hecker wrote a whitepaper, citing Raymond’s *The Cathedral and the Bazaar* in an attempt to persuade Netscape executives to “open source” their web browser.⁹⁰ On 4 February 1998, Raymond was invited by Netscape for a strategy conference at Silicon Valley. Six days after the conference (on 10 February), Raymond revised *The Cathedral and the Bazaar* again by adding an “Epilogue” (in Revision 1.31) that

⁸⁸ Bezroukov points out that the level of decentralisation in the Bazaar model may have been exaggerated by Raymond. Especially, the core team of Linux have quite more power than the peripheral contributions: “The black and white picture painted in CatB (monolithic, authoritarian Cathedral model vs. democratic, distributed Bazaar model) is too simplistic. These metaphors for high centralization (Cathedral) and no centralization (Bazaar) do not account for the size of a given project; its complexity, timeframe and time pressures; its access to resources and tools; and, whether we are talking about core functionally (like Linux kernel) or peripheral parts of the system. For large projects like operating systems it is especially important that the core of the system is developed in a highly centralized fashion with a small core team. Peripheral parts of the system can benefit from a more relaxed, more decentralized approach”. See Nikolai Bezroukov, “A Second Look at the Cathedral and the Bazaar”, (1999) 4 *First Monday* 12, at

<http://firstmonday.org/issues/issue4_12/bezroukov/index.html>

Bezroukov, “Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism),” (1999) 4 (10) *First Monday* at <http://www.firstmonday.org/issues/issue4_10/bezroukov/index.html#b4>;

⁸⁹ “Netscape Announces Plans to Make Next-Generation Communicator Source Code Available Free on the Net” at <<http://wp.netscape.com/newsref/pr/newsrelease558.html>>

⁹⁰ Raymond’s paper was actually written seven months before Netscape’s announcement to go “open source” in January 1998. Before that point, Raymond was not personally involved with Netscape’s decision to be an “open source” company. See Jim Hamerly and Tom Paquin with Susan Walton, “Freeing the Source—The Story of Mozilla”, *Open Sources: Voices from the Open Source Revolution* eds. by Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O’Reilly & Associates, 1999) pp.197-8

fanfared Netscape's shift as "a large-scale, real-world test of the bazaar model in the commercial world":

The open-source culture now faces a danger; if Netscape's execution doesn't work, the open-source concept may be so discredited that the commercial world won't touch it again for another decade. On the other hand, this is also a spectacular opportunity. Initial reaction to the move on Wall Street and elsewhere has been cautiously positive. We're being given a chance to prove ourselves, too. If Netscape regains substantial market share through this move, it just may set off a long-overdue revolution in the software industry.⁹¹

In late February, Raymond, along with Bruce Perens (the then-leader of the Debian project⁹²), co-founded the Open Source Initiative (OSI) and produced the Open Source Definition (OSD). The purpose of the OSI is to monitor and facilitate the use of the OSD by software projects. The OSD was not written from scratch. Earlier in 1997, the Debian Community produced the *Debian Free Software Guideline* and Perens rehashed this guideline into the OSD by just leaving out the Debian-specific references in it.⁹³ The OSD itself is not a license but a list of common elements that could be adopted by any collaborative "open source" project and its corresponding licence. It specifies ten common criteria against which a software project can be found to be "open source" or not. According to these ten criteria, an open "source project" must 1) allow free distribution of software, 2) make source code publicly available, 3) allow modifications and derived works, 4) ensure integrity of the author's source code, 5) allow no discrimination against persons or groups, 6) allow no discrimination against fields of endeavour, 7) require no signature to accept the licence, 8) be not specific to a certain product, 9) allows no "contamination" of other software distributed on the same medium and 10) be technology-neutral in licensing software.⁹⁴ Compared with Stallman's Free Software Definition (FSD)⁹⁵, the OSD obviously contains a more detailed and specific list about what an "open source"

⁹¹ Raymond, *The Cathedral*, supra note 14

⁹² The Debian project was managed by a global team of volunteers aiming to produce an operating system distribution that was "composed entirely of free software." See Debian, *A Brief History of Debian* at <<http://www.debian.org/doc/manuals/project-history/ch-intro.en.html>>

⁹³ The Debian guideline is available at <http://www.debian.org/social_contract#guidelines>

⁹⁴ The last criterion was later amended in 2004 in response to the increasing popular use of click-wrap licences. See the OSD at <<http://opensource.org/docs/definition.php>>

⁹⁵ Stallman, "The Free Software Definition", at <<http://www.gnu.org/philosophy/free-sw.html>>

project and its corresponding licence scheme should be. However, the basic licensing principles of the OSD do not drastically deviate from the spirit of the FSD. Especially, the first three criteria of the OSD, which list some defining features of open source software, are not drastically dissimilar from the requirements specified in FSD.

-OSD Criterion (1) Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

-OSD Criterion (2) Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

-OSD Criterion (3) Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.⁹⁶

Perens hails the OSD as the “bill of rights for the computer users” because it defines “certain rights that a software license must grant [users] to be certified as Open Source.”⁹⁷ He distils these rights under the OSD into three principles:

- The right to make copies of the program, and distribute those copies.
- The right to have access to the software’s source code, a necessary preliminary before you can change it
- The right to make improvements to the program.⁹⁸

The above open source principles are almost identical with the FSD (except that the wording is slightly different). Stallman also observes that the OSD is only “derived

⁹⁶ The rest of seven criteria are: “4. Integrity of The Author's Source Code; 5. No Discrimination Against Persons or Groups; 6. No Discrimination Against Fields of Endeavor; 7. Distribution of License; 8. License Must Not Be Specific to a Product; 9. License Must Not Restrict Other Software; 10. License Must Be Technology-Neutral.”OSI, Open Source Definition, at <<http://www.opensource.org/docs/osd>>

⁹⁷ Perens, “The Open Source Definition”, Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O'Reilly & Associates, 1999) p.171

⁹⁸ *ibid.*, p.172

indirectly from” the rules set by the FSD, because FSD is also focused on the protection of software users’ “rights” (though in the FSD they are called “freedoms”) in these respects.

It is interesting to note there are two “labels” that can be applied to the same type of non-proprietary software after 1998—“free software” and “open source”—both of which allow free access, modification and redistribution. The Linux project would continue to be “free software” since its inception, but it could also be called “open source” software after 1998. More importantly, Linux used the same licence—the GNU GPL—before and after 1998. Kelty points out an irony that the advocates of “free software” and proponents of “open source” seem to enter into a debate over something upon which they practically agree: “the creation of two names allowed people to identify *one thing*, for these two names referred to identical practices, licenses, tools, and organizations”:

Free Software and Open Source shared everything “material,” but differed vocally and at great length with respect to ideology. Stallman was denounced as a kook, a communist, an idealist, and a dogmatic holding back the successful adoption of Open Source by business; Raymond and users of “open source” were charged with selling out the ideals of freedom and autonomy, with the dilution of the principles and the promise of Free Software, as well as with being stooges of capitalist domination. Meanwhile, both groups proceeded to create objects—principally software—using tools that they agreed on, concepts of openness that they agreed on, licenses that they agreed on, and organizational schemes that they agreed on. Yet never was there fiercer debate about the definition of Free Software.⁹⁹

So if the two labels refer to the same kind of software (and the same type of software licences as well), what is the real consequence of Raymond’s “open source” movement? Does it only introduce a distinction without a difference? Does the invention of “open source” really alter the course where “free software” would have gone after 1998?

⁹⁹ Kelty, *Two Bits*, p.117

My answer is that “open source” movement is more than a matter of changing label, but there have been at least two real impacts on the “free software” movement. The twofold impacts operate mutually on both movements: “open source” expands the influence of “free software” beyond the computer hacker community, while “free software” puts an “ethical” limit on how far this expansion can go. Firstly, the “open source” campaign significantly expands the reach of the “free software” and puts it firmly into the consciousness of the general public. Especially, it raises substantial awareness about the commercial potential of “free software” in the business world. It convinces quite a significant part of the conventional non-hacking world that the decentralised Bazaar model of mass collaboration “to the point of promiscuity” can be employed to produce high-quality software. The post-1998 “free software”, after being rebranded as “open source”, was no longer a monastic hacker subculture subsisting on the lingering Hacker Ethic, but it became increasingly in vogue among corporate executives and salespeople. Of course, this newly acquired popularity is achieved by toning down Stallman’s strong political language about the full commitment to “software freedom”.

The second impact of “open source” is that it challenges Stallman to further defend the ultimate value of “free software” more rigorously in order to put an “ethical” limit on the otherwise unbridled commercialism of open source. Different from merely having a few knee-jerk reactions to some specific incidents in the 1980s (such as in Xerox and Symbolics incidents in the early 1980s¹⁰⁰), Stallman after 1998 needed to make conscious effort to clarify the “ethical” underpinning for the latter-day Hacker Ethic. The most obvious example of these efforts is embodied in Stallman’s article—“Why Open Source Misses the Point of Free Software”—which is written as a direct response to the challenge posed by “open source”. In this article, he stresses that there is a pronounced difference between “free software” and “open source” in terms of the message about “freedom” that was intended to be sent or kept quiet about:

Nearly all open source software is free software. The two terms describe almost the same category of software, but they stand for views based on fundamentally

¹⁰⁰ See Section 2.3.2 of this chapter.

different values. *Open source is a development methodology; free software is a social movement.* For the free software movement, free software is an *ethical imperative*, because only free software respects the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software “better”—in a practical sense only. It says that nonfree software is an inferior solution to the practical problem at hand. For the free software movement, however, nonfree software is a social problem, and the solution is to stop using it and move to free software.¹⁰¹ (added emphasis)

In the same article, Stallman further clarifies that software freedom needs to be guarded as an intrinsic value for its own sake. The measurement resides solely in the four kinds of software freedom given to software users, but not the performance of software. “Open source” is not necessarily a superior “development methodology” and it may well produce lower-quality software than the “closed source” software model. Open source advocates miss this point by wrongly believing that “open source” software is guaranteed to be more “powerful” and “reliable” than any proprietary one.¹⁰² The reality is that it is possible for proprietary software to outperform non-proprietary software. Without a strong belief in the intrinsic value of “software freedom”, users can easily be lured away by some practical advantages offered by proprietary software.

Sooner or later these users will be invited to switch back to proprietary software for some practical advantage. Countless companies seek to offer such temptation, some even offering copies gratis. Why would users decline? Only if they have learned to value the freedom free software gives them, to value freedom in and of itself rather than the technical and practical convenience of specific free software. To spread this idea, we have to talk about freedom. A certain amount of the “keep quiet” approach to business can be useful for the

¹⁰¹ Stallman, “Why Open Source Misses the Point of Free Software” at <<http://www.gnu.org/philosophy/open-source-misses-the-point.html>>

¹⁰² Stallman argues: “The idea of open source is that allowing users to change and redistribute the software will make it more powerful and reliable. But this is not guaranteed. Developers of proprietary software are not necessarily incompetent. Sometimes they produce a program that is powerful and reliable, even though it does not respect the users' freedom.” *ibid.*

community, but it is dangerous if it becomes so common that the love of freedom comes to seem like an eccentricity.¹⁰³

However, Stallman's insistence that "open source is a development methodology; free software is a social movement" is not universally accepted. His argument tends to attract two kinds of objection. The first kind argues that "open source" is more qualified as a consciously organised "social movement" than "free software" is. Stallman's free software campaign before 1998 could be seen as no more than an outlet of a lonely hacker's frustration in reaction to a series of unhappy incidents.¹⁰⁴ In contrast, "open source" was consciously started as a movement in 1998 and it immediately attracted a lot of supporters who had already been working on leading "open source" projects such as Linux, Sendmail, Perl, Python, Apache. The official history of the Open Source Initiative documents that after the meeting organised by Tim O'Reilly on 8 April 1998, all the above participating "open source" programmers "voted to promote the use of the term 'open source', and agreed to adopt with it the new rhetoric of pragmatism and market-friendliness that Raymond had been developing."¹⁰⁵

The second objection argues that it is futile to find whether "open source" or "free software" is a "movement" (or two "movements"). What really matters is the fact that "open source" and "free software" programmers share the same "platform", which invites and encourages debate and discussion over issues pertinent to software development. This is the view held by Kelty, who calls this "platform" a "recursive public". In this public, unmediated discourses are freely exchanged among programmers through their human language as well as their technological language in source code. This public is also said to be a "recursive" one, because Kelty believes that it dissolves the traditional distinction of software being a *technical* system and the organisation of software programmers being a *social* system. In this sense, FOSS "recurses" through the *technical* and *social* dimensions into one single "self-grounding" public sphere, which operates independently from other established

¹⁰³ *ibid.*

¹⁰⁴ The term "free software" was used in the *GNU Initial Announcement* as early as in 1983 when the FSD did not exist. It would take some more years before Stallman had a clearer view about free software as defined in the FSD. See for more detail in Section 2.3.2 of this chapter.

¹⁰⁵ See OSI, "History of the OSI", at <<http://www.opensource.org/history>>

social structures such as the price mechanism of the market and the for-profit agenda under a corporate structure.¹⁰⁶ (In contrast, most commercial proprietary software developers tend not to have such free-following “discourses” in a self-grounding public, because their *technical* output is mostly likely to be dictated or incentivised by either market or firms.) It is also important to note that the recursive public is neither a formal organisation nor an aimless crowd, but it sits somewhere between. Kelty believes that “recursive public” is a better term than “movement” to describe the unique phenomenon of “free software” and “open source” as sharing *one and the same* “public” platform:

Free Software and Open Source are neither corporations nor organizations nor consortia (for there are no organizations to consort); they are neither national, subnational, nor international; they are not “collectives” because no membership is required or assumed—indeed to hear someone assert ‘I belong’ to Free Software or Open Source would sound absurd to anyone who does. Neither are they shady bands of hackers, crackers, or thieves meeting in the dead of night, which is to say that they are not an “informal” organization, because there is no formal equivalent to mimic or annul. Nor are they quite a crowd, for a crowd can attract participants who have no idea what the goal of the crowd is; also, crowds are temporary, while movements extend over time. It may be that *movement* is the best term of the lot, but unlike social movements, whose organization and momentum are fueled by shared causes or broken by ideological dispute, Free Software and Open Source share practices first, and ideologies second. It is this fact that is the strongest confirmation that they are a recursive public, a form of public that is as concerned with the material practical means of becoming public as it is with any given public debate.¹⁰⁷

In other words, it does not matter whether “free software” and “open source” is one or two “social movements” or none at all, but what is most significant is that they share the same debating platform, which is the same “recursive public”. I largely agree with Kelty that the debates between “open source” and “free software” are essential to create one shared “recursive public” between the two camps. However, I

¹⁰⁶ Kelty, *Two Bits*, pp.10-11

¹⁰⁷ Kelty, *Two Bits*, p.113

am also afraid that Kelty does not pay enough attention to the fact that it was the “Hacker Ethic” (and its latter-day versions, i.e., FSD and OSD) that tied this public together in the first place. One should not take for granted the happening of the recursive public, and this public can never dispense with a minimum consensus as agreed by “open source” and “free software” partisans on basic tenets in the Hacker Ethic. Unfortunately, Kelty is rather dismissive of the Hacker Ethic’s role in guiding programmers’ behaviour and its relevance in the recursive public: “While hackers themselves might understand the hacker ethic as an unchanging set of moral norms, their practices belie this belief and demonstrate how ethics and norms can emerge suddenly and sharply, undergo repeated transformations, and bifurcate into ideologically distinct camps (Free Software vs. Open Source), even as the practices remain stable relative to them.”¹⁰⁸ As has been shown above, from the Hacker Ethic to “free software” to “open source”, the *core* stewardship obligations to protect software freedom as indicated in Tenets (1) and (2) of Levy’s Hacker Ethic have largely reincarnated in Stallman’s FSD and Raymond’s OSD. The Hacker Ethic, as well as its spirit in various FOSS licences, provides a minimum consensus between “open source” and “free software” programmers. It glues programmers from both camps together to collaborate towards common projects. In other words, the Hacker Ethic is the common ground that underpins the collaborative practice of open source and free software. Without this minimum consensus, Kelty’s “recursive public” would not exist in the first place let alone survive the ever-changing socio-legal environment of software production. In short, although “open source” adds a commercial twist to the non-proprietary software movement in the third historical stage, it does not fundamentally change the “glue”, i.e. the Hacker Ethic, which brings together the collaborative efforts that build common software projects.

2.5 Conclusion

This chapter has surveyed a brief history of the Hacker Ethic since its inception in the relatively close-knit computer hacker community in the 1950s and 1960s, followed by the decline of the Hacker Ethic in the late 1970s and then the non-proprietary software movement in an attempt to revive the lost Hacker Ethic from the

¹⁰⁸ *ibid.*, p.180

late 1980s onwards. In particular, Stallman invented a copyleft licensing scheme that for the first time verbalised programmers' minimum commitment that is intended to support large-scale radically decentralised collaborative software projects. In 1998, the "open source" campaign led by Raymond openly broke away from Stallman's "free software" movement to form a business-friendly group that has the ambition to succeed on the mainstream commercial software market, but the underlying Hacker Ethic of the movement has remained largely unchanged. In the next chapter, I will examine how the FOSS programmers find their legal expression of software freedom through "intellectual property" licensing schemes in some detail.

Chapter 3 Intellectual Property and Software Freedom

3.1 Introduction

After the 1980s it gradually transpired that the rise of intellectual property (IP), especially copyright and patent, in software became an increasingly influential factor affecting FOSS communities. The impact of IP was twofold. Firstly, it weakened hacker custom as a great number of computer hackers were lured away to write the more lucrative proprietary software. However, secondly, a small number of stalwart software freedom fighters puzzled out that copyright licences could be drafted in a way to continue the threatened custom. The first impact has mainly been covered in the previous chapter. It is the job of this chapter to explore how computer hackers attempt to reconfigure the IP system through their licensing schemes (e.g. GPL) in order to reinstate the principles of software freedom for their community.

The chapter is divided into five parts. The first part (Section 3.2) explores Richard Stallman's argument against using the umbrella term "intellectual property" that conventionally lumps together a set of disparate bodies of law, mainly including copyright and patent laws. As FOSS licences are attentive to the subtle differences between copyright and patent as well as their respective impact on FOSS collaboration, it is necessary for this chapter to examine these two areas separately. The second part (Section 3.3) is a general introduction to software copyright law as the background against which FOSS licensing schemes are crafted. It discusses some major developments in Anglo-American copyright law that have positively or negatively affected software freedom. The third part (Section 3.4) explores the rise of "software patents" as a response to the IP expansionists' failure to stretch copyright further to cover the non-expressive elements, i.e. functionality of software since the 1990s. It canvasses the debate about the patentability of computer programs in both EPC countries (especially UK) and the US. I will show how FOSS programmers perceive patents as a threat to software freedom. The fourth part (Section 3.5) uses the GPL as an example to show how principles of software freedom are articulated mainly through the language of copyright law. It also examines how the GPL partially contains the perceived threat to the hacker ethic from patents. The fifth part

(Section 3.6) concludes that FOSS programmers do not endorse “IP” as a unified body of law but they selectively leverage two different branches of “IP” (i.e. copyright and patent) to protect software freedom in a non-exclusive commons regime.

3.2 “Intellectual Property” and FOSS

An exploration of “software freedom”, which is a necessary condition of commons-based decentralised collaborative programming, ironically has to “start[] with the other side of the coin, property rights”.¹ This is largely because the idea of “software freedom” in FOSS licensing was first triggered as the computer hackers’ response to the rise of intellectual property in software programs.² It reflects what Houweling calls the “property turn” in the FOSS movement that has run concurrently with a broader movement known as “cultural environmentalism”.³ The employment of FOSS licences is not drastically dissimilar to environmentalists’ efforts to enlist the property regime to impose land obligations such as the much used “conservation easement”.⁴ The “property turn” embodied in FOSS licensing indicates that a FOSS *commons* is different from the *public domain* that is a property-free zone. FOSS programmers do not relinquish their IP rights altogether, but they rearrange the initial entitlements as conferred by IP law. Along this line, Boyle differentiates two kinds of freedom as institutionalised respectively in the public domain and the licensing commons (the commons produced as a result of FOSS licensing): “In the public domain, freedom is based on the absence of property rights. In the licensing commons, freedom is based on the preemptive *exercise* of the property rights by the rights holder in order to grant use privileges to users of the commons, and sometimes

¹ Lawrence Rosen, *Open Source Licensing—Software Freedom and Intellectual Property Law*, (Upper Saddle River, NJ: Prentice Hall PTR, 2005) p.13

² See Chapter 2 for detail.

³ See Molly Shaffer Van Houweling, “Cultural Environmentalism and the Constructed Commons”, (2007) 70 *Law and Contemporary Problems* 23 at 29-33

⁴ For “conservation easement” in the context of environmental protection, see, for example, Nancy A. McLaughlin, “Rethinking the Perpetual Nature of Conservation Easements”, (2005) 29 *Harvard Environmental Law Review* 421

to bind those future users to add their own improvements back to the common pool.”⁵ (original emphasis)

Although a licensed FOSS commons is built upon the institution of IP, it would be inappropriate to leap to the conclusion that FOSS programmers embrace wholeheartedly the idea of “property” in general and that of “IP” in particular. The reality is a bit more complex than that in two respects. First, the use of licensing schemes mainly reflects the pragmatic side of the FOSS movement. Licensing is not intended to, and cannot be, a complete overhaul of the existing IP system, but they are workarounds or makeshift solutions to particular defects of the legal system as identified by FOSS programmers.⁶ Not unlike computer hackers’ “patches” or “bug fixes” that are designed to fix some particular problems in a software program, FOSS licences are the equivalent of “computing hacks” in the legal world. They are privately made legal “patches” submitted to plug holes in publicly made IP law based on FOSS programmers’ diagnoses. For example, Kelty believes that GPL is exactly Stallman’s “hack” into the US IP regime.⁷ However, this legal pragmatism of “patching” and “hacking” should not obscure the idealistic side of the FOSS movement, which attempts to reverse the programming environment back to a situation similar to the pre-1980s IP-free hacker community.⁸ Although it is uncertain whether this ideal of creating an IP-free zone can be realised in the near future, it at least reminds us that the current arrangements under various FOSS licensing schemes are largely a compromise between private property and the hacker ethic.

Secondly, FOSS programmers are aware that the so-called “intellectual property law” is not a unified body of law, but conventionally has at least three major sub-sets,

⁵See James Boyle, “Cultural Environmentalism and Beyond” (2007) 70 *Law and Contemporary Problems* 5 at 10

⁶There is no shortage of suggestion that more changes should be done through legislative route by amending IP laws rather than private ordering through licensing schemes. See for example, Severine Dusollier, “Sharing Access to Intellectual Property Through Private Ordering”, (2007) 82 *Chicago-Kent Law Review* 1391 at 1435

⁷Kelty write: “The GNU General Public License (GPL), written initially by Richard Stallman, is often referred to a beautiful, clever, powerful ‘hack’ of intellectual-property law—when it isn’t being denounced as a viral, infectious object threatening the very fabric of economy and society.” Kelty, *Two Bits*, p.179

⁸Bear in mind the top commandment on the agenda of the free software movement is the “[a]bolition of all forms of private property in ideas.” See Moglen, *The dotCommunist Manifesto*, January 2003, <<http://emoglen.law.columbia.edu/publications/dcm.html>>

comprising copyright, patent and trademark.⁹ Stallman is famous for his persistent refusal to use the umbrella term “intellectual property”, and he argues that “IP” is merely a “seductive mirage” that does not exist in reality.¹⁰ It would be misleading to lump these three disparate categories together as if they are a unified whole, because each of them respectively plays quite a different role in FOSS licensing: Very briefly, FOSS licences rely primarily on *copyright*, which protects software programs as if they are literary works. To *patent* software is hugely controversial in the Anglo-American world and hard-core free software programmers are normally against the use of patents. In order to protect their goodwill and reputation, it is not unusual nowadays for FOSS programmers to seek *trademark* protection for indicators of the origin of their projects and associated products or services. In short, it would be an inappropriate question to ask whether FOSS programmers are *for* or *against* “IP”, but it is necessary to have a more nuanced approach by examining separately the roles of copyright, patent and trademark in FOSS licensing.

Furthermore, it is also important to note that Stallman’s rejection is closely linked with his criticism of mainstream economic (or in Stallman’s parlance, simplistic “economistic”) thinking behind the term “IP”:

The term “intellectual property” also leads to simplistic thinking. It leads people to focus on the meager commonality in form that these disparate laws have—that they create artificial privileges for certain parties—and to disregard the details which form their substance: the specific restrictions each law places

⁹ Apart from these three sub-areas of IP, software is also commonly protected as trade secrets through non-disclosure agreements. However, FOSS by definition has its source code freely available to the public, so trade secrecy is not an issue here and thus is not discussed in this chapter. For the use of confidentiality rules to protect software programs in the UK context, see David Bainbridge, Chapter 11, *Legal Protection of Computer Software* (Heywards Heath, West Sussex: Tottel Publishing, 2008, 5th Ed.) pp.321-339; For a US perspective on this issue, see for example, Gregory J. Maier, “Software Protection—Integrating Patent, Copyright and Trade Secret Law”, (1987) 69 *Journal of Patent and Trademark Office Society* 151 at 162-5

¹⁰ Stallman, “Did You Say ‘Intellectual Property’? It’s a Seductive Mirage” at <<http://www.gnu.org/philosophy/not-ipr.html>>; Stallman’s view can be contrasted with the lawyers’ convention of taking for granted the term “IP” in the context of software licensing. For example, Rosen argues that software is “a product of human intellect, and therefore it is a kind of *intellectual property*. Intellectual property is a valuable *property* interest, and the law allows its owner to possess and control it. The programmer who writes software—or the company that hires that person to write software—is deemed to be the first owner of intellectual property embodied in that software. That owner may exercise dominion over that intellectual property”. (original emphasis) See Rosen, *Open Source Licensing*, supra note 1, p.14

on the public, and the consequences that result. This simplistic focus on the form encourages an “economistic” approach to all these issues.¹¹

Economistic thinking, he goes on, serves as “a vehicle for unexamined assumptions” where only the quantity of software production matters, but “freedom and way of life do not.” With this in mind, I will now examine copyright and patent in turn to see how they are respectively viewed by FOSS programmers. As trademarks have more to do with the FOSS programmers’ manifestation of their collective authorship in collaborative projects, they are not dealt with in this chapter but will be analysed later in Chapter 5 which is dedicated to the issue of FOSS authorship.

3.3 Copyright and FOSS

From the late 1970s onwards, developments in statutory and case laws in the Anglo-American world gradually established copyright as the main mode of legal protection for computer programs. In the US, the 1978 final report prepared by the National Commission on New Technological Uses of Copyrighted Works (CONTU) recommended that copyright should be extended to software. This recommendation was enacted by the 1980 amendment of the US 1976 Copyright Act that expressly included “computer program” as a subject matter. In this amended Act, a computer program is defined as “a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result” and is protected as a kind of literary work.¹² The copyright scholar Melville Nimmer, in his capacity as Vice Chairman of the CONTU, clarified that the existing general copyright principles should in a wholesale fashion be applied to software programs just like any other copyright subject matter:

CONTU did not recommend, and did not intend, any change in the continuing applicability to programs of general copyright principles—e.g., as to the copyrightability and infringement—in effect following the enactment of the general revision of the Copyright Act in 1976. The general copyright principles applicable to programs have been, and remain, those which are applicable to

¹¹ Stallman, “Did You Say ‘Intellectual Property’? It’s a Seductive Mirage”, *ibid.*

¹² 17 U.S.C. 101

novels, plays, directories, dictionaries, textbooks, musical works, maps, motions pictures, sound recordings, and other categories of works.¹³

In the UK, the Copyright (Computer Software) Amendment Act 1985 for the first time specifically included software programs in the “literary work” category under the 1956 Copyright Act. The subsequent 1988 Copyright Design Patent Act (CDPA) also provides that copyrights subsist in software programs as “literary works”. Section 3 (1b), defines “literary work” as “any work, other than a dramatic or musical work, which is written, spoken or sung, and accordingly includes [...] a computer program [...]”¹⁴ Unlike the US copyright law, the CDPA does not have a definition for “computer program”, which arguably has the advantage of being flexible to include new technologies such as HTML programs.¹⁵

3.3.1 The Originality Threshold

Copyright law requires that programs be original to merit protection. Anglo-American copyright law does not set a very high threshold for “originality”, but it is not always an easy task to ascertain the degree of “originality” that qualifies a piece of code for copyright subsistence. In the US copyright subsists in “original works of authorship”¹⁶ and a work is “original” in the sense that it is “independently created by the author (as opposed to copied from other works), and that it possesses at least *some minimal degree of creativity*”.¹⁷ (added emphasis) In the UK the threshold is arguably even lower, with no explicit requirement of a work to be minimally creative. Copyright may subsist in a work as long as it is not copied from other human-made sources and is a result of the author’s own skill, judgment or labour.¹⁸ In contrast, the European continental legal tradition tends to have a more demanding requirement of

¹³ Melville Nimmer, “Declaration” in *Appendix* to “Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs” by Anthony L. Clapes, Patrick Lynch and Mark R. Steinberg (1987) 34 *UCLA Law Review* 1493

¹⁴ UK CDPA, 1998; According to Section 3 (1c), “preparatory design material for a computer program” is also protected as “literary work”.

¹⁵ See Stanley Lai, *The Copyright Protection of Computer Software in the United Kingdom* (Oxford and Portland, Oregon: Hart Publishing, 2000) p.14

¹⁶ 17 U.S.C. 102

¹⁷ *Feist Publication Inc. v. Rural Telephone Service Inc.* (1991) 499 US 340, 345

¹⁸ *University of London Press Ltd. v University Tutorial Press Ltd.* [1916] 2 Ch 601

originality for works including computer programs.¹⁹ In an attempt to harmonise national differences among countries in Europe, Article 1(3) of 1991 EU Software Directive gives a definition of “originality” as follows:

A computer program shall be protected if it is original in the sense that it is the *author’s own intellectual creation*.²⁰ (added emphasis)

The Directive further makes it clear that the author’s “intellectual creation” is the sole criterion of copyright subsistence and “[n]o other criteria shall be applied to determine its eligibility for protection.”²¹ Unfortunately, the UK draftsman responsible for preparing the implementing regulations assumes that the existing UK copyright originality standard has already been practically compatible with the Directive’s definition of originality as the “author’s own intellectual creation” and there was no need to change the wording in the corresponding section of UK copyright law. Lai suspects that this assumption may well not be true.²² In case of conflict, Bainbridge argues, the Directive’s requirement of originality for computer programs should prevail over the English one: “It is beyond doubt that a judge in the United Kingdom would apply [Directive’s] test rather than the traditional view of judges of what originality meant, even if a common thread could be determined.”²³

There can be three types of “original” copyrightable contributions arising from a FOSS project. First, if the code is completely written from scratch by contributors for the project, it is highly likely to pass any of the three aforementioned tests of originality for having a “minimal degree of creativity” (US), or using programmers’ “skill, judgement and labour” (UK), or being authors’ “own intellectual creation” (EU). Secondly, copyright may also subsist in modifications of preexisting code if

¹⁹ Lai observes that, for example, German courts “did not only require individuality as compared with pre-existing programs, but also that the ability shown in the engineering process considerably surpassed average programming ability”. Stanley Lai, *supra* note 15, p.17

²⁰ Council Directive 91/250, 1991 O.J. (L122)

²¹ *ibid.*

²² For example, “many modern computer programs are effectively compilations of standard modules”. So by the UK standard, they are original but by the Directive’s standard they may well not be so. See Lai, *supra* note 15, FN 38, p.17

²³ Bainbridge has a slightly different view from Lai’s as shown above, and he believes programs that pass the UK test would be unlikely to fail to pass the Directive’s test: “[...] most computer programs, unless trivial or made up of a selection of commonly known or public domain elements requiring no skill or judgement in their selection or arrangement, will be considered to be intellectual creations.” See David Bainbridge, *Legal Protection of Computer Software* (Heywards Heath, West Sussex: Tottel Publishing, 2008, 5th Ed.), p.64

these modifications are “original” enough to be recognised by copyright law. In the US, a copyrightable modification can be a “derivative work” which means the work “based upon one or more preexisting works [...] or any other form in which a work may be recast, transformed, or adapted.”²⁴ Similarly in the UK, the modified code may give rise to a fresh copyright if it passes the minimum threshold of originality.²⁵ As software is also a functional artifact, there is a domino effect, where one tiny modification may lead to a series of follow-up changes in order to make the whole program operate properly. Bainbridge observes that even if a single modification is too trivial to be original, the many modifications together may cumulate to qualify for a fresh copyright:

Where the modifications represent the author’s own intellectual creation, a fresh copyright will be created in the new version of the program. It may be that this applies to an accumulation of numerous modifications, each of which in themselves might not reach the standard for originality. However, even making a small modification to a computer may require the exercise of a great deal of skill as the programmer has to check that the modification works correctly and that the effect it has on the existing and retained parts of the program is as intended. This can call for a significant amount of testing and further modification. Even a small modification can have unpredictable consequences and end up involving far more work than originally envisaged.²⁶

That a fresh copyright is recognised as subsisting in modifications or derivative works is crucial to peer-produced FOSS projects. This is because FOSS collaboration is built upon incremental creativity by many collaborative programmers rather than a single breakthrough invention by the initial creators. Rosen points out that copyright ownership in FOSS can be seen as *a chain of title*, where “[a]n original work of authorship is the first link in the chain” and this “chain is elongated during the collaborative open source development process.”²⁷ The strength of this chain of title in many follow-up modifications can be a measure of the robustness of the collaborative relations in a collaborative FOSS project. In other words, the maturity

²⁴ 17 U.S.C. ss.101 & 103

²⁵ See *Ibcos Computers Ltd. v. Barclays Mercantile Highland Finance* [1994] FSR 275

²⁶ Bainbridge, *supra* note 23, pp. 65-66

²⁷ Rosen, *Open Source Licensing*, *supra* note 1, p.28

of a FOSS project can be roughly shown by the length of the chain of title: “Mature open source projects often consist of software passed through many such stages of aggregation and modification, their original works of authorship proudly displaying a long chain of title including the names of many individuals and organizations that preceded them.”²⁸ I will show later that GPL is important exactly because it makes sure the “chain of title” in a FOSS project is unbroken by imposing the copyleft condition on distributing the GPL covered code.

Thirdly, there can be a “compilation” copyright that subsists in the aggregated work comprising all submitted contributions to a FOSS project as a collective whole. The UK copyright protects “compilation” as a kind of “literary work”,²⁹ which is different from the term “database” as defined in the CDPA³⁰. Bainbridge argues that a software program can be a “compilation” but not a “database”, because the former is an undivided collective whole while the latter comprises separable independent works: “Where a computer program is made up of individual modules, those modules cannot be described as independent. They work together as a whole application. Therefore, the whole may have a separate copyright as compilation independent of any copyright in the modules as programs in their own right.”³¹ In the US context, the term “compilation” has a slightly different meaning than that in the UK. The US “compilation” copyright covers “collective works” where “a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole.”³² (Note that the US “collective works” cover “separate and independent” works, which are different from Bainbridge’s understanding of software as “compilation” comprising inseparable interdependent modules in the UK context.³³) The copyright in software as a “collective work”,

²⁸ *ibid.*, p.29

²⁹ s. 3 (1) (a) CDPA 1988

³⁰ Note that the CDPA has explicitly incorporated the European standard of originality for “database” copyright: a database is “original” if it constitutes “the author’s own intellectual creation.” s.3 A, CDPA 1988

³¹ Bainbridge, *supra* note 23, p.67

³² 17 U.S.C. 101

³³ In fact, Bainbridge’s understanding of software as “compilation” works under the CDPA is probably closer to the meaning of “joint work” in the US context. The US copyright law defines “joint work” as “a work prepared by two or more authors with the intention that *their contributions be merged into inseparable or interdependent parts of a unitary whole.*” (added emphasis) 17 U.S.C. 101; For the difference between “collective works” and “joint works” for software, see Rosen, *supra* note 1, pp. 32-33

according to Rosen, is “a reflection of the originality of the collection and its organizational structure rather than of the individual components. Most software is a copyrightable collection of modules. The arrangement and organization of the collection of individual modules are often the most original aspects of a software program.”³⁴ The recognition of originality in “compilation” or “collective works” has a largely positive impact on FOSS projects, whose design of the modular “architecture”³⁵ prizes the lead developers’ creative efforts in aggregating individual contributions into a coherent collective *whole*. These efforts of aggregation are not necessarily mindless mechanical work but they involve selecting, testing, and approving (and sometimes declining) code sent by contributors,³⁶ and they are highly likely to be beyond the threshold of originality. This is especially true for the Linux kernel project, where Torvalds and his fellow subsystem maintainers have devoted themselves to aggregating a huge amount of peer-produced contributions into a coherent whole. In this light, Eric Raymond and Catherine Raymond strongly advise that it would be beneficial for FOSS project leaders to always register copyright in their project as original “collective works” with the copyright registration authority in the US.³⁷

In summary, although the threshold of originality to qualify for copyright subsistence is low, it does exist for computer programs. All sustained FOSS projects would contain a huge number of contributions with various degree of originality. The biggest problem that FOSS projects face is not about whether contributed code is “original” enough to attract copyright protection. Most contributions will easily pass the originality threshold individually on their own merit. On top of this, these contributions together will also cumulatively give rise to copyright in “compilations” or “collective works”. The really difficult problem that needs to be tackled is that

³⁴ Rosen, *supra* note 1, p.27

³⁵ Pumfrey J suggests that there could be originality in the software “architecture” itself, which was likened to the plot of a play: “It seems to be generally accepted that the ‘architecture’ of a computer program is capable of protection if a substantial part of the programmer’s skill, labour and judgement went into it. In this context ‘architecture’ is a vague and ambiguous term. It may be used to refer to the overall structure of the system at a very high level of abstraction.” *Cantor Fitzgerald International v Tradition (UK) Ltd.* [2000] RPC 95 at 134

³⁶ Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update* at <<http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf>>

³⁷ Eric Raymond, Catherine Olanich Raymond, *Licensing HOWTO* (9 November 2002) at <<http://catb.org/~esr/Licensing-HOWTO.html>>

many original contributions will form a huge network of ownership interests by many copyright holders. It is not always an easy task to coordinate these many ownership interests for the purpose of building one coherent project and the copyright system will not automatically splice them together. In this scenario, FOSS licences step in to solve the problem of coordination by standardising the legal commitments of many copyright holding contributors. These licences make peer-produced contributions legally compatible with each other in a decentralised environment.

3.3.2 Software as Expression and Function

It is understandable that to analogise a computer program to a literary work³⁸ has the advantage of fitting software as a new technological form into an existing copyrightable subject matter, but this is not an entirely accurate analogy. Software is not ordinary literary text written and read by human beings, but, more importantly, it contains instructions that operate computerised functions. In short, software has a dual nature of being both *expressive* like literal texts and *functional* like machines.³⁹ Recall that software is written in source code by programmers and it can be compiled into object code that can be executed by computers. On the one hand, the human-readable source code is just like any other form of human *expression* such as novels, speech scripts or sheet music scores. On the other hand, the machine-readable object code turns software into *functional* artifacts that instruct computers to “manipulate symbols leading to virtual or physical effects, such as making calculations, displaying information on a screen, controlling the path of a cutting device or an industrial process.”⁴⁰ This dual nature of software as expression and function is well reflected in Laddie *et. al.*’s definition of software “program” in the UK context (while the CDPA does not define what is software): a software program is “a series of instructions capable of being fed to a computer system, by typing in at a keyboard

³⁸ Internationally, it has also become settled that software—including both source code and object code—are subject to copyright protection as “literary works”. According to the 1994 Agreement on Trade Related-Aspect of Intellectual Property Rights (TRIPS), “[c]omputer programs, whether in source or object code, shall be protected as literary works under the Berne Convention (1971)”. Article 10 (1), TRIPS Agreement 1994

³⁹ Martin Kretschmer “Software as Text and Machine: The Legal Capture of Digital Innovation”, 2003 (1) *The Journal of Information, Law and Technology* (JILT) at <http://www2.warwick.ac.uk/fac/soc/law/elj/jilt/2003_1/kretschmer/>

⁴⁰ David Bainbridge, *Legal Protection of Computer Software* (Haywards Heath, West Sussex: Tottel Publishing, 2008) p.53

or in any other way, and, when so entered, of controlling its operation in a desired manner.”⁴¹

As a general rule, copyright protects expressions, but not functions, of software: “it is a programmer’s *expression* of some functionality that may be protected by copyright, and not the functionality itself.”⁴² (original emphasis) Unfortunately the water has already been muddied in reality, partially because it is not always easy to separate functionality neatly from expression in computer programs.⁴³ There is no shortage of attempts by proprietary software developers to broaden copyright protection to cover functionality of software. Since the mid-1980s, there has been a series of cases concerning whether or not copyright protection could be stretched to give protection to the “non-literal” or “non-textual” (i.e. functional) elements in software on both side of the Atlantic. In the US, the Court of Appeal for Third Circuit, in the 1986 landmark case *Whelan Associates v. Jaslow Dental Laboratory*, ruled that “even absent copying the literal elements of the program” the defendant infringed the copyright in the non-textual “structure” of a record-keeping program by the plaintiff.⁴⁴ This ruling effectively stretched copyright protection to the non-expressive part of the software program. The *Whelan* decision was much criticised for giving the overbroad protection to software⁴⁵ but it is welcomed, mainly by IP expansionist commentators, as a way of compensating for the lack of clear patent protection of the functionality of software programs in the mid-1980s. For example, Maier argues that the *Whelan* court reached an equitable result during a time when the US legal system was extremely uncertain about whether software-related inventions could get patent protection:

In effect, copyright protection has been stretched in *Whelan* to fill the gap left when the courts denied software inventions patent protection. Stretching

⁴¹ Hugh Laddie, Peter Prescott, Mary Vitoria, Adrian Speck, and Lindsay Lane, *The Modern Law of Copyright and Designs* (London, Edinburgh & Dublin: Butterworth, 2000, 3rd edition) Vol. 2, p.1610

⁴² Software Freedom Law Center, “Originality Requirements under U.S. and E.U. Copyright Law”, 27 September 2007, at <<http://www.softwarefreedom.org/resources/2007/originality-requirements.html>>

⁴³ Dan Burk, “Copyrightable Function and Patentable Speech” (2001) 44 (2) *Communications of the ACM* 69

⁴⁴ 797 F.2d 1222 (3d Cir. 1986) at 1234

⁴⁵ For example, Lemley criticised the *Whelan* decision for sacrificing “accuracy in separating protectable from unprotectable material in order to achieve a workable rule that is easy to apply.” Mark Lemley, “Convergence in the Law of Software Copyright”, (1995) 10 *High Technology Law Journal* 1, at 12

copyright protection is understandable, from an equitable point of view, to protect software authors/inventors who were discouraged from seeking patent protection due to the changing status of the law regarding the patentability of software inventions. The equities are particularly important in cases involving misconduct. Prospectively, however, as the intellectual property community accepts the notion that software is patentable, there may ultimately be little need to so stretch the bounds of copyright protection.⁴⁶

The expansionist rationale in *Whelan* made its way into a few subsequent cases including the highly controversial *Lotus Development Corporation v. Paperback Software International*, where Judge Keeton decided that the menu command hierarchy in the Lotus 1-2-3 spreadsheet program was protected by copyright.⁴⁷ It is important to know that although Lotus Development Corporation won this case, it failed, two years later, to secure copyright protection for the same non-literal menu system in a second *Lotus* case against another company.⁴⁸ The first *Lotus* ruling is generally regarded as the high-water mark of copyright protection of the “look and feel” or the user interfaces in software.⁴⁹ Free software programmers reacted strongly against this expansionist tendency. Shortly after Lotus filed the first lawsuit against Paperback, Richard Stallman and his followers organised a mass picket outside the Lotus headquarters to publicise the danger of giving software companies overbroad protection of their software products.⁵⁰

The *Whelan* and the first *Lotus* decisions follow the so-called “broad constructionist” approach of software copyright, because they have broadened the reach of copyright to cover the non-expressive part of software. This “broad constructionism” is contended by the “narrow constructionism” that believes copyright should be limited to textual copying of software:

⁴⁶ Gregory Maier, *supra* note 9, at 161

⁴⁷ 740 F. Supp. 37 (D Mass, 1990).

⁴⁸ This second *Lotus* case went further to the US Supreme Court. As Justice Stevens did not vote, the rest of the eight justices reached a 4:4 decision, leaving the First Circuit decision unchanged. *Lotus v. Borland*, 49 F.3d 807 (1st Cir. 1992); 516 US 233 (1996)

⁴⁹ For the significance of *Lotus v. Paperback*, see Lai, *supra* note 15, pp.68-70

⁵⁰ See the *infra* text accompanying the notes 57-62

On one side, ‘broad constructionists’ have emphasized the need to compare the copyrighted and accused works as a whole, in order to give protection to the ‘total concept and feel’ of the works. On the other side, ‘narrow constructionists’ have urged the methodical dissection of copyrighted works into their component parts in order to determine what exactly qualifies for copyright protection.⁵¹

In 1992, things started to change when the Second Circuit made a landmark “narrow constructionist” ruling in *Computer Associates v. Altai*, which greatly reined in the “broad constructionist” tendency in US software copyright. In this case, Judge Walker devised a much more nuanced three-step test by 1) dissecting software into different levels of abstraction, 2) filtering out non-protectable elements and 3) comparing the remaining core expressive parts to see if there is a substantial similarity between the program alleged to have been infringed and the allegedly infringing program.⁵² This abstraction-filtration-comparison *Altai* test has been followed in later US cases and it has also spread to non-US jurisdictions.⁵³ For example, in the UK context, the use of *Altai* test was endorsed by Ferris J. in *Richardson v. Flanders*⁵⁴ but was later rejected by Jacob J., who in *Ibcos v. Barclays* favoured using the indigenous English test of the “overborrowing of skill, labour and judgement which went into the copyright work”.⁵⁵ Lai worries that the English test lacks prescriptive precision to guide future cases involving non-literal copying of software. Jacob’s rejection of the *Altai* test, which could have usefully filtered out non-protectable elements, might lead to a *Whelan*-type overprotection of software in the UK: “Due to the absence of a prescriptive test/criterion [similar to the *Altai* test], UK software copyright law will be placed in a more invidious position than the US, if *Ibcos* is followed. Arguably, the scope of software copyright protection is

⁵¹ Lemley, “Convergence in the Law of Software Copyright”, supra note 45 at 2

⁵² 982 F.2d 693 (2d Cir.1992) at 707-712

⁵³ In the US, the test is further refined in a Tenth Circuit’s decision, which divides a program into six levels of declining abstraction comprising 1) the general purpose 2) structure or architecture 3) modules 4) algorithms and data structures, 5) the source code, 6) the object code. The last two levels are the most detailed and least abstract expressions of a program. *Gates Rubber Co. v. Bando Chemical Industries*, 9 f.3d 823 (10th Cir. 1995) at 835; For the spread of *Altai* test outside US, see Lai, supra note 15, FN 137, p.30

⁵⁴ [1993] FSR 497

⁵⁵ [1994] FSR 275 at 302

presently greater in the United Kingdom than in the USA. *Whelan*-type fears of broad protection are there to be realised for the future.”⁵⁶

Campaign against Copyright Protection of Non-literal Element of Software

After the 1980s, FOSS programmers generally accepted copyright subsistence in the expressive part of software. However, they reacted strongly against extending copyright protection further to the non-literal part, especially the “look and feel”, of computer programs. When Lotus Development Corporation brought its first copyright lawsuit to protect the menu system of their Lotus 1-2-3 spreadsheet program, the free software community were deeply worried that this move would jeopardise the creative freedom that software programmers would be allowed to have in the future.⁵⁷ On 24 May 1989, Stallman and two other prominent computer scientists (including Marvin Minsky, the founder of the MIT Artificial Intelligence Lab⁵⁸) orchestrated a large-scale demonstration against the Lotus’s “look and feel” copyright litigation. Over two hundred people, most of whom were MIT professors and students, marched from the MIT campus to join a rally outside the Lotus headquarters based at Cambridge, Massachusetts. They carried placards bearing signs such as “Creative companies don’t need to sue” and “Oh no! Look and feel copyright!” and chanted a hexadecimal protest slogan:

Hey, hey, ho, ho, software tyranny has got to go
1-2-3-4, toss the lawyers out the door
5-6-7-8, innovate don’t litigate
9-A-B-C, 1-2-3 is not for me
D-E-F-O, look and feel have got to go⁵⁹

It is important to know that the protest was not just targeted at Lotus, but more broadly it registered the computer hackers’ growing unease about the copyright

⁵⁶ Lai, *supra* note 15, p.49

⁵⁷ The then vice president and general counsel of Lotus, Tom Lemberg argued that “[t]he copyright law we believe is absolutely essential to the health of this entire industry” and “[i]t is the means for several centuries now to reward creations.” See Jane FitzSimon, “MIT Software Developers Field ‘Freedom’ Campaign: Apple, Lotus ‘Look-and-Feel’ Suits Targeted In Ad”, *The Boston Globe* available at <<http://www.skytel.co.cr/advocacy/research/1989/0424.html>>

⁵⁸ Minsky is also a featured hacker in Steven Levy’s book. He was said to be a “[p]layful and brilliant MIT prof who headed AI lab and allowed the hackers to run free.” Levy, *Hackers*, p.11

⁵⁹ League for Programming Freedom, “Programmers and Users Picket Lotus, Protesting User-Interface Copyright Litigation” 24 May 1989, at <<http://www.skytel.co.cr/advocacy/research/1989/0524-b.html>>

expansionist trend as represented by a series of similar undergoing copyright disputes over the non-literal copying of graphic user interfaces.⁶⁰ It is feared that by giving protection to the “look and feel” of software, other programmers would be effectively stopped from independently writing their own code to achieve identical functions. Stallman likens the consequence to that of giving monopoly over steering a car: “If there were copyrights like this on cars, then every manufacturer would have to give you a different way to steer [...] If you learned to drive a Ford, you wouldn't know how to drive Chevrolets. Some cars would have throttles, others would have joysticks, and each manufacturer would have to find a new way of doing it”.⁶¹ With the introduction of the “look and feel” copyright, it is a slippery slope where software programmers would be deprived of the freedom to mimic the non-literal aspect of other programmers’ software even though they do not involve the act of literal copying.

The high turnout at the Lotus protest is a sign of the lingering impact of the old Hacker Ethic that originated from the MIT AI lab but was challenged by proprietary software in the late 1970s. It shows that the Hacker Ethic was not quite dead in the late 1980s and the early 1990s, and it still played a role in forging solidarity among programmers who are against copyright expansion. The most enthusiastic anti-Lotus protestors, in late 1989, formed the League for Programming Freedom (LPF), also under the leadership of Stallman, with an aim “to prevent monopolies on software development”.⁶² As the second *Lotus* case closed the door to the “look and feel” copyright completely (i.e., graphic user interface copyright could no longer pose any further threat to programming freedom)⁶³, the LPF shifted to another battlefield,

⁶⁰ For example, Stallman’s protest was also prompted by an earlier lawsuit brought by Apple against Microsoft. Stallman commissioned a designed badge showing a fanged Apple logo with a serpent body, indicating Apple’s aggressiveness in getting copyright protection of its user interface. LPF,



“The History of the LPF” at <<http://www.progfree.org/History/history.html>>

⁶¹ Alan Cooperman, “Scientists Challenge Companies’ Lock on Software Programs” 25 May 1989 at <<http://groups.csail.mit.edu/mac/projects/lpf/Links/prep.ai.mit.edu/demo.ap-wire>>

⁶² LPF, “History of the LPF”, supra note 60

⁶³ *Lotus v. Borland* 49 F.3d 807 (1st Cir. 1992); 516 US 233(1996)

where they campaigned hard against the growing monopolistic effect of software patents.⁶⁴

3.3.3 Scope of Exclusivity: Restricted and Permitted Acts

Software programmers, as the original authors of their works, have some exclusive rights to do certain restricted acts in relation to their programs. However, these exclusive rights do not amount to the software authors' total and absolute ownership of their works, and they are normally subjected to various exceptions mandated by copyright law. These exceptions in effect narrow the scope of exclusivity by allowing non-owners to do certain acts without the original programmers' permission. In the US, the Copyright Act 1976 gives copyright holders five exclusive rights 1) to make copies, 2) to prepare derivative works, 3) to distribute copies of the original work or derivative works, 4) to perform certain kinds of works publicly and 5) to display certain kinds of works.⁶⁵ Specific to software, there are two important limitations on these exclusive rights. Firstly, users are allowed to make a copy or adaptation of the computer program "as an essential step in the utilization" of it.⁶⁶ Secondly, users are also allowed to make back-up copies of the program for archival purposes.⁶⁷ In other words, lawful computer users can do these two acts without permission from software authors.

In the UK, the copyright holders have a slightly different list of exclusive rights to do the certain acts "restricted" by copyright. They are the exclusive rights (a) to copy the work; (b) to issue copies of the work to the public; (ba) to rent or lend the work to the public; (c) to perform, show or play the work to the public; (d) to communicate the work to the public; (e) to make an adaptation of the work or do any of the above in relation to an adaptation.⁶⁸ Outside the purview of these exclusive rights, there is also a host of general "permitted acts" that can be done without a copyright holder's permission.⁶⁹ Specific to computer programs, there are four important "permitted

⁶⁴ See infra Section 3.4 for detail.

⁶⁵ 17 USC 106

⁶⁶ For example, a lawful user can copy a program to a disk or memory when this is essential to run the program on a computer. 17 U.S.C. 117 (a) (1)

⁶⁷ However, all archival copies need to be destroyed "in the event that continued possession of the computer program should cease to be rightful". 17 U.S.C. s.117 (a) (2)

⁶⁸ s.16 (1), CDDA 1988

⁶⁹ ss. 29-31 CDDA 1988

acts” that further narrow the scope of programmers’ exclusive rights. It is not an infringement of copyright for a lawful user to 1) make back up copies of a program⁷⁰; 2) to decompile the program to achieve interoperability⁷¹; 3) to observe, study and test the functioning of the program⁷²; 4) to copy or adapt the program necessary for his lawful use especially for the purpose of correcting errors in the program.⁷³ It is important to note that it is not possible to contract out of the first three exceptions by means of licensing agreements. Under s. 294A(1) CDPA, licensing terms that forbid lawful users from doing these three permitted acts are unenforceable:

Where a person has the use of a computer program under an agreement, any term or condition in the agreement shall be void in so far as it purports to prohibit or restrict—

- (a) the making of any back up copy of the program which it is necessary for him to have for the purposes of the agreed use;
- (b) where the conditions in section 50B(2) are met, the decompiling of the program; or
- (c) the observing, studying or testing of the functioning of the program in accordance with section 50BA.⁷⁴

However, it is possible to contract out of the fourth exception, which allows copying or adaptation of a program when necessary for the purpose of its “lawful use”.⁷⁵ This permitted act is sometimes seen as an equivalent of the “non-derogation from grant” doctrine making its way into UK software copyright law.⁷⁶ According to Lord Temple, this doctrine means “that a grantor will not be allowed to derogate from his grant by using property retained by him in such a way as to render property granted by him unfit or materially unfit for the purpose for which the grant was made [...]”.⁷⁷ Section 50C of the CDPA codifies this doctrine by preventing software copyright

⁷⁰ s.50A, CDPA

⁷¹ s.50B CDPA (implementing EU Software Directive Article 6 on decompilation for the purpose of achieving “interoperability” between programs)

⁷² s.50 BA, CDPA 1988

⁷³ s.50C(1) (2) CDPA

⁷⁴ 296A, CDPA

⁷⁵ “It is not an infringement of copyright for a lawful user of a copy of a computer program to copy or adapt it, provided that the copying or adapting - (a) is necessary for his lawful use; and (b) is not prohibited under any term or condition of an agreement regulating the circumstances in which his use is lawful.” s. 50C (1) CDPA

⁷⁶ Bainbridge, *supra* note 23, p.94

⁷⁷ *British Leyland Motor Corp v. Armstrong Patents Co Ltd* [1986] AC 577 at 641. For this doctrine applied in copyright in general, see also, Andrew Shindler, “Derogation from Grant in Copyright Law” (1986) 49 (4) *Modern Law Review* 513

holders from imposing unnecessary restrictions that would otherwise defeat the purpose of the lawful use of the software in the absence of a contrary agreement.

In the spirit of Section 50C, it is worth noting that the CDPA does give lawful software users a highly circumscribed right to debug software. Section 50C(2) stipulates that it may “be necessary for the lawful use of a computer program to copy it or adapt it for the purpose of correcting errors in it” provided that there is no agreement to the contrary. It is understandable that most proprietary software companies would contract out of this permitted act by not giving users the right to correct errors by themselves in the licensing schemes. This is because these companies have “a vested interest in providing on-going maintenance, including error correction, to their licensees”.⁷⁸ However, independent from this circumscribed right to debug, it is sometimes speculated that software users may resort to the “right to repair” (or the “spare parts” exception) created in *British Leyland v. Armstrong* as an analogous device to achieve the identical purpose of correcting errors. In *British Leyland*, the majority opinion of the House of Lords decided that customers had a “right to repair” that overrode car manufacturers’ copyright in the drawings of its car exhaust system.⁷⁹ This “right to repair” was later confirmed in the software case *Saphena v. Allied Collection*, where the defendant was held to be entitled to modify the provided source code in order to debug the software.⁸⁰ Unfortunately, despite the *Saphena* decision, it is highly uncertain whether *British Leyland* could always be cited as a firm authority for software users to claim their right to debug software. This is because the “right to repair” as invented in *British Leyland* is a doctrinally unsound policy decision, and it has been criticised for being an unsatisfactory product of “a blatant piece of judicial legislation” that overrode the statutory exclusive right of a copyright owner.⁸¹ Bainbridge argues that this “right to repair” is unlikely to play an important role in limiting software owners’ exclusive rights:

On balance, it is difficult to say with any certainty that the *British Leyland* right to repair can apply to error correction of computer programs. There will usually

⁷⁸ See Bainbridge, *supra* note 23, pp.95-6

⁷⁹ [1986] AC 577

⁸⁰ It is worth noting that although the court recognised that that the defendant had a “right to repair” but it did not have the right to *improve* the software beyond fixing the bugs. [1995] FSR 616

⁸¹ See Laddie *et. al.*, *The Modern Law of Copyright and Designs*, Vol. 2 pp.2254-5

be an expectation that the software company will correct errors and, in the vast majority of cases, a maintenance agreement will be entered into by the parties. It is often part and parcel of the original agreement. The right to repair may be available in limited cases such as where the software company is no longer willing or able to correct errors but its wider application is doubtful.⁸²

It is not difficult to find that a wide scope of exclusive rights that have been given to software authors by copyright is not really conducive to FOSS collaboration based on the peer-production model. In order to build up a large-scale FOSS project, “peers” must work in an environment where each other’s code can be readily reproduced, modified, debugged and redistributed on a frequent basis. The copyright regime, by default, seems disproportionately skewed towards the economic interests of proprietary software developers who have little intention to collaborate with software users. It assumes that software programs are discrete products that are mainly developed by software programmers in isolation from the outside world and at the same time these programmers’ efforts must be rewarded by exclusive property rights. This assumption is too simplistic to account for many collaborative non-proprietary software programming activities that do not rely on exclusive property rights. In this light, FOSS licences are designed to squeeze the broad scope of exclusive rights by copyright owners in order to create a software commons suitable for decentralised collaborative programming. Under these licences, programmers voluntarily relinquish almost all of their exclusive rights and everyone is invited to freely “copy” or “adapt” each other’s code. Perens’s three principles distilled from the Open Source Definitions illustrate how software users are empowered by having three “rights” to software under FOSS licensing schemes: they have “1) The right to make copies of the program, and distribute those copies. 2) The right to have access to the software’s source code, a necessary preliminary before you can change it. 3) The right to make improvements to the program.”⁸³ These three rights substantially expand the scope of software users’ “permitted acts” than are initially allowed by

⁸² Bainbridge, *supra* note 23, p.97

⁸³ Bruce Perens, “The Open Source Definition” in *Open Sources: Voices from the Open Source Revolution* eds. by Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O’Reilly & Associates, 1999) p.172

copyright. I will come back to this issue through an examination of GPL in some detail in *infra* Section 3.5.1.

3.4 Patent and FOSS

There are two possible routes to get legal protection for the *functions* of software by intellectual property law.⁸⁴ The first route is to stretch copyright to cover the non-literal elements of software as what was achieved in early software copyright cases such as *Whelan*⁸⁵, while the second route is to patent software as computer-implemented inventions. As is discussed above, the first route now has been blocked in the US since the *Altai* decision introduced the abstraction-filtration-comparison test to disqualify the non-expressive part of software for copyright protection.⁸⁶ Software developers who are keen to offset the effect of the *Altai* decision now have to go down the second route by patenting their works. This second route is favoured by IP expansionists who are interested in creating a seamless protection spectrum where the post-*Altai* copyright regime is supplemented by the patent system. Maier's argument for "a unique continuum of intellectual property protection" of software is representative of this view:

One must not suppose that copyright and patent protection are in any way at odds. Copyright protection can mesh very neatly with patent protection to provide a unique continuum of intellectual property protection in the software environment. Copyright protects against literal copying and against slavish imitation of code or mode of expression. Patent protects against infringing use, whether through derivation or independent development, of the broader functional aspects of software thus the combination of available copyright and patent protection would appear to make software the most protectable of all technology [...].⁸⁷

It is not difficult to see that Maier's argument in favour of a maximalist protection under the copyright-patent continuum is one-sidedly presented from a profit-

⁸⁴ As is discussed in the preceding sub-section, after the *Altai* ruling, it has been largely settled that copyright only protects the *expression*, but not the *function*, of software.

⁸⁵ *Whelan v. Jaslow* 797 F.2d 1222 (3d Cir. 1986)

⁸⁶ 928 F.2d 693 (2d Cir.1992)

⁸⁷ Maier, *supra* note 9 at 161

maximising perspective held by some, if not all, commercial proprietary software developers.⁸⁸ In contrast to Maier's view, many FOSS developers, especially those hard-core free software proponents, are vehemently against software patents, which are believed to be the potential and actual threat to FOSS projects. Free Software Foundation (FSF) and League for Programming Freedom (LPF), both of which are under the leadership of Stallman, are among the most vocal voices calling for abolishing "software patents".⁸⁹ However, it would be wrong to make a sweeping statement that FOSS is an antithesis to software patents. Not all FOSS developers wish to abolish software patents, but some of them take a more reconciliatory position that the patent system could be reformed. This reformist view is mainly held by corporate open-source participants, who are financially better-resourced, to defuse patent infringement allegations and even to build their own defensive portfolios of patents. This bifurcation of patent abolitionism and reformism is indicative of a growing schism between the camp of "pure" volunteer contributors and that of corporate contributors. Although Stallman has never been against corporate participation of FOSS projects, he draws the line at the issue of software patents. The following two subsections will delve into the impact of patents on software freedom by examining 1) the legal meaning of "software patents" and 2) the considerable controversy caused by these patents in relation to FOSS.

3.4.1 Patentability of Software-Related Inventions

Strictly speaking, there is no such thing as "software patent", because software or computer programs standing alone or "as such" are normally excluded from being a patentable subject matter under the Anglo-American patent law. In fact, the term "software patent" is often merely used in a loose sense and it does not really have an agreed-upon legal meaning. Software Freedom Law Centre's (SFLC), in their official guide advising patent defences for FOSS developers, deliberately choose to avoid this term:

⁸⁸ Not all software businesses are pro-patent, but the reality is a bit more complex than what has been suggested by Maier. It is observed that, unlike the pharmaceutical industry that tends to have a dominant consensus that "vigorous patent enforcement is the best policy", the software industry simply lacks such a consensus. See John R. Allison, Abe Dunn and Ronald J. Mann, "Software Patents, Incumbents, and Entry", (2007) 85 *Texas Law Review* 1579

⁸⁹ Preamble, GPL 3.0; LPF (Gordon Irlam and Ross William), *Software Patents: An Industry at Risk*, 1994, at <<http://www.progfree.org/Patents/industry-at-risk.html>>

[...] we avoid use of the term ‘software patent,’ which has no generally agreed-upon definition. Under current U.S. law, software per se is (probably) not patentable, but it is generally a simple exercise in artful legal drafting to represent a software-related invention as a claim covering patentable subject matter (generally by reciting generic, well-known hardware features). Although the details differ, the basic situation is much the same in many other countries, despite a widely-held misconception in the FOSS community that the patentability of software-related inventions is peculiar to U.S. law.⁹⁰

This thesis prefers to use the term “software-related invention” (or simply “software invention”)⁹¹ or “computer-implemented invention” (CII)⁹², either of which is slightly more accurate in reflecting the actual state of affairs. The reason behind this preference is as follows: what is under the heated “software patent” debate concerns not the easy case of clearly unpatentable software *as such*, but the more complicated case of the alleged “inventions” employing “software” as a component. Because the legal boundary of these software inventions is not always clear-cut, its wide reach may well profoundly affect FOSS collaborative projects. However, this terminological preference for “software-related invention” should not be read as a call for categorically banning the use of “software patents” in the literature. To the contrary, it is intended to give a clearer picture of what much-discussed “software patent” as a legal phenomenon is really about and why it is so strongly opposed to by hard-core abolitionist free software campaigners. As patent laws about software-related inventions are not exactly the same in the UK (and within the bigger context of European Patent Convention) and the US, I will explain the two patent regimes separately. This explanation will set the scene for a critical understanding of the

⁹⁰ SFLC (Richard Fontana et. al.), *A Legal Issues Primer for Open Source and Free Software Projects*, 3 March 2008, at <<http://www.softwarefreedom.org/resources/2008/foss-primer.html>> FN3, p.21

⁹¹ Bainbridge defined “software invention” as “an invention within a range of inventions which are implemented by means involving or including a programmed computer.” See Bainbridge, *supra* note 23, p.284

⁹² This term is used by the proposed Directive on patentability of CII, which is defined as “any invention the performance of which involves the use of a computer, computer network or other programmable apparatus and having one or more *prima facie* novel features which are realised wholly or partly by means of a computer program or computer programs.” In 2002, this Directive, as an attempt to codify the case law of the EPO, was proposed but it was rejected by the European Parliament in 2005 due to the lack of consensus among member countries. However, the term CII is still used in the scholarly literature. For example, Bainbridge believes that “CII” is synonymous with the term “software invention” as is defined by himself. *ibid.*

debate between the “software patent” abolitionists and reformists within the FOSS community.

UK and the EPC Regime: Interpreting “Technical Character”

In the UK, the statutory language makes it a clear rule that a computer program “as such”, however innovative it may be, cannot be a patentable “invention” as defined by the Patents Act (PA) 1977⁹³. This rule is the localisation of the Article 52 of the European Patent Convention (EPC), though its wording fails to adopt the official English text of the EPC.⁹⁴ Under the EPC, whether a subject matter is patentable depends upon the definition of “invention” provided by the EPC: “European patents shall be granted for any inventions, *in all fields of technology*, provided that they are new, involve an inventive step and are susceptible of industrial application.” (added emphasis)⁹⁵ It is noteworthy that the text “in all fields of technology” was later inserted to the original wording of the EPC 1973, in order to synchronise the European patent system with the requirement in Art. 27 (1) TRIPS Agreement 1994⁹⁶. Very importantly, Art 52 (2) EPC narrows the meaning of “invention” as in Art. 52(1) by making a list of unpatentable subject matters including “programs for computers”. So far the rule seems to be reasonably clear that computer programs are excluded from the meaning of “invention” and are thus unpatentable, but the third paragraph in the Art 52 would cause much confusion and eventually lead to a divergence of opinion on software-related “inventions” between the UK Court of Appeals and that of the European Patent Office (EPO). This Art 52 (3) is often known as the “as such” proviso and it reads:

[The EPC] excludes the patentability of the subject-matter or activities referred to therein *only to the extent to which a European patent application or European patent relates to such subject-matter or activities as such.* (added emphasis)

⁹³ ss.1 (1) & (2), UK Patents Act 1977

⁹⁴ In particular, the EPC was amended in 2000 to synchronise with Art. 27 of the TRIPS Agreement by adding patents should be allowed in “all fields of technology”. This change has not been reflected in the PA either.

⁹⁵ Art 52 (1)

⁹⁶ Art. 27 (1) reads: “[...] patents shall be available for any inventions, whether products or processes, *in all fields of technology*, provided that they are new, involve an inventive step and are capable of industrial application.” (added emphasis) The UK PA has not incorporated this phrase so far.

Far from spelling an end to the debate over the patentability of software, this “as such” proviso simply invites more confusion and demands further interpretation of its meaning. It is believed that the text of Art. 52 turns out to be the source of “the ongoing uncertainty of the scope of the exclusion from patentability of computer programmers” and the meaning of “as such” is “anyone’s guess during the past two decades”.⁹⁷ After the mid-1980s, in an attempt to give some level of certainty to the meaning of “invention” in relation to software under the EPC, the Technical Boards of Appeal (TBA) of the EPO made a series of decisions focusing on whether a subject matter has the necessary “technical character” to be a patentable “invention”.⁹⁸ Among these cases, TBA’s 1987 landmark decision on *Vicom/Computer-related invention*, where a method of processing digital images was examined, stands out as the one of most significance. In *Vicom*, TBA established the famous “technical contribution” test: “Decisive is what technical contribution the invention as defined in the claim when considered as a whole makes to the known art”.⁹⁹ In other words, a claimed subject matter would not be patentable if it fails to make a non-obvious “technical contribution” to the known art. This *Vicom* test is important because it sets the scene for the TBA to interpret the “technical character” that qualifies the subject matter to fall under the meaning of “invention” under Art. 52, and it was also later adopted by the UK Court of Appeal in *Merrill Lynch’s Application*¹⁰⁰ and *Gale’s Application*¹⁰¹. A variant of the *Vicom* test later made its way into the final text of the abortive EU Directive on Computer-Implemented Inventions, where the “technical contribution” was expressly required for patentability: “Member States shall ensure that it is a condition of involving an inventive step that a computer-implemented invention must make a *technical contribution*.”¹⁰² (added emphasis)

⁹⁷ Noam Shemtov, “Software Patents and Open Source Models in Europe: Does the FOSS Community Need to Worry about Current Attitudes at the EPO?” (2010) 2 (2) *International Free and Open Source Software Law Review* 151 at 156

⁹⁸ For the chequered history of EPO’s decisions on the meaning of “invention” and the patentability of CII in the 1980s, see Justine Pila, “Dispute over the Meaning of ‘Invention’ in Art. 52(2) EPC—The Patentability of Computer-Implemented Inventions in Europe” (2005) 36 *IIC* (2) 173 at 174-6

⁹⁹ EPO Board of Appeal, T208/84 (1987)

¹⁰⁰ [1989] RPC 561

¹⁰¹ [1991] RPC 305

¹⁰² Art. 4 EU Directive on CII

Unfortunately, the TBA, in the following years, gradually drifted away from its own technical contribution test used in *Vicom*, towards a more expansive interpretation of “technical character”. The abandonment of the “technical contribution” test did not happen at one stroke but it started almost imperceptibly and then unfolded through “a series of incremental changes without express disapproval of *Vicom*” by the TBA.¹⁰³ Three cases, i.e. PBS Partnership/Pension Benefits System¹⁰⁴, Hitachi/Auction Method¹⁰⁵ and Microsoft/Clipboard Format I&II¹⁰⁶, are often singled out to show a trajectory of TBA’s gradual deviation from *Vicom* to embrace the new “any hardware” test. Note that the old *Vicom* test is actually an “inventive step” test in disguise because a patentable subject matter must make a non-obvious technical contribution to the known art in the first place. In contrast, the new “any hardware” test eliminates this built-in “inventive step” requirement. If the claim is made to a physical apparatus, it will be considered to be a patentable subject matter, regardless of whether this “invention” makes “technical contribution” to the known art. The “any hardware” test substantially expands the meaning of “technical character” and thus lowers the patentability threshold, which moves ever-closer to the removal of the statutory prohibition of patenting software as such under Art. 52 (2) and (3).¹⁰⁷

The EPO’s embrace of the “any hardware” approach has caused both confusion and frustration to the UK Court of Appeal, which struggles to stick to the “technical contribution” approach adopted by the its own binding precedents such as *Merrill Lynch’s Application*¹⁰⁸ and *Gale’s Application*¹⁰⁹. In *Aerotel Ltd. v. Telco Holdings Ltd*, Jacob L.J. argues that it becomes very difficult for the English court to be perfectly in keeping with the recent development of EPO’s ever-changing jurisprudence on patentability of software-related inventions. He finds that the TBA does not follow its own precedents rigorously but it has come up with six different interpretations of “technical character” of a patentable “invention” (including three variants of the “any hardware” test), none of them are consistent with each other among themselves. The UK court has no choice but to follow its own precedents by

¹⁰³ Bainbridge, supra note 23, p.295

¹⁰⁴ Case T931/95 [2002] EPOR 522

¹⁰⁵ Case T258/03 [2004] EPOR 548

¹⁰⁶ Case T424/03 [2006] EPOR 39; Case T411[2006] EPOR 40

¹⁰⁷ Bainbridge, supra note 23, p.295

¹⁰⁸ [1989] RPC 561

¹⁰⁹ [1991] RPC 305

using the more onerous technical contribution test due to the doctrine of *stare decisis*.¹¹⁰ The upshot of the *Aerotel* decision is that the UK insists on a higher patentability threshold than the current standard used by the EPO.¹¹¹

To summarise, there has been great definitional uncertainty surrounding the meaning of patentable “invention” as defined by the Art. 52 of the EPC. The EPO has tried to reduce the uncertainty by pegging the issue to the meaning of “technical character”, which turns out to be equally difficult to pin down. Although the EPO has failed to produce a consistently used test to determine the “technical character” of a claimed subject matter, it has the tendency to gradually stretch the elastic reach of “technical character” and thus lower the patentability threshold over the years. It has also led to an unfortunate divergence between the EPO and the UK Court of Appeal on this issue.

United States: From *Benson* (1972) to *Bilski* (2010)

On the other side of the Atlantic, the US law governing the patentability of software-related inventions has no less a chequered history than its European counterpart. Section 101 of the US Patent Act 1952 defines patent-eligible subject matters as “any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof”.¹¹² However, unlike the EPC, this Act does not contain a statutory list of non-“inventions” that are unpatentable. So what is actually excluded from patent-eligible subject matter relies on the US case law to fill the gap. A well-accepted list of exceptions to patentable matters can be found in a leading Supreme Court decision including “laws of nature, natural phenomenon, and abstract ideas”, which can be roughly seen as an equivalent of Art 52 (2) of the EPC in the US context.¹¹³

¹¹⁰ [2007] 7 RPC 117; In a more recent UK case *Symbian Ltd. v Comptroller-General of Patents*, the Court of Appeal, though using a less hostile tone to the EPO’s case law, confirms that technical contribution test should be retained in the UK. [2008] Bus. L.R. 607

¹¹¹ Bainbridge finds that this is “regrettable” result , because the more onerous UK test would drive inventors to apply for software-related invention patents in the EPO rather than the UK IPO. Bainbridge, *supra* note 23, pp.315-6

¹¹² 35 U.S. 101

¹¹³ *Diamond v. Diehr* 450 U.S. 175, 185 (1981)

Neither “software” nor “computer program” is explicitly mentioned in the statutory text of Section 101 of the Patent Act. It has remained extremely uncertain as to whether “software” or “computer program” falls under any of the four categories of “process, machine, manufacture, or composition of matter” defined by Section 101, until three leading cases were decided by the Supreme Court between the early 1970s to the early 1980. The first of them is *Gottschalk v. Benson* in 1972. The Supreme Court ruled that a computer program using a mathematical algorithm to convert binary-coded decimal numbers into pure binary numerals was not a patentable subject matter under Section 101.¹¹⁴ The *Benson* approach was confirmed by the 1978 decision in *Parker v. Flook*, where a computer program using another algorithm to control alarm limits in catalytic conversions of petrochemicals was again ruled to be unpatentable.¹¹⁵ The final case, arguably the most significant of the three, is *Diamond v. Diehr* decided in 1981. In *Diehr*, the court had to tackle the question as to whether a computerised process of curing raw synthetic rubber that employed a well-known mathematical formula known as “Arrhenius Equation” was a patentable subject matter under Section 101. The court ruled a “mathematical formula *as such* is not accorded the protection of patent laws, and this principle cannot be circumvented by attempting to limit the use of the formula to a particular technological environment.”¹¹⁶ (added emphasis) However, although a mathematic formula “as such” is not patentable, when it is tied to a special-purpose computer, the claimed subject matter “as a whole” will pass the patentability threshold according to the *Diehr* court.¹¹⁷ In other words, the *Diehr* decision is a refinement of the previous court rulings. It has the effect of making computer program “as such” unpatentable, which makes the US law start to bear some resemblance to the law under Art 52 (2) & (3) EPC.¹¹⁸

In the next thirty years after *Diehr* the US Supreme Court did not again hear any case on the patentability issue until the 2010 *Bilski* decision.¹¹⁹ This unfortunately has

¹¹⁴ 409 U.S. 63, 71-2 (1972)

¹¹⁵ 437 U.S. 584 (1978)

¹¹⁶ 450 U.S. 175 at 191

¹¹⁷ *ibid.*, at 188

¹¹⁸ Furthermore, it is also thought that *Diehr* in the US and *Vicom* in Europe play similar historical role in making software-related inventions patentable. See Martin Kretschmer, “Software as Text and Machine: The Legal Capture of Digital Innovation”, *supra* note 39

¹¹⁹ *Bilski v. Kappos* 130 S. Ct. 3218 (2010)

created an opportunity for a lower patent court and the US patent office to gradually drift away from the holding in *Diehr*. During the intervening period between *Diehr* and *Bilski*, the task of adjudicating on the patentability disputes moved from the Supreme Court completely to a lower-level specialist patent court—Court of Appeals of Federal Circuit (CAFC)—which was established one year after *Diehr* in 1982 with an attempt to achieve some level of uniformity in enforcing US patent law at the federal appellate level. In 1998 the CAFC ruled that a computerised business method could be patented in *State Street Bank & Trust v. Signature Financial Group*. Disregarding previous Supreme Court’s rulings on the patentability requirement, the CAFC created its own test that a claimed subject matter, with no need to have any particular physical embodiment, would be patentable so long as it produced a “useful, concrete and tangible result”.¹²⁰ Some commentators believe that the *State Street* ruling effectively removes the threshold of patentability in the US between 1998 and 2008, during which Section 101 of US Patent Act became “a dead letter”.¹²¹ During these ten years, US became a place that was very generous to issue patents to software-related inventions.¹²²

However, in 2008, the CAFC thought it was a time to rein in the proliferation of patents as a result of the *State Street* decision. In *Bilski v. Kappos*, where a business method of hedging financial risk were examined, the CAFC discontinued its own “useful, concrete and tangible result” test used in *State Street*. Resuming the line of jurisprudence developed by the Supreme Court in *Benson*, *Flook* and *Diehr*, the CAFC decided to go back to Section 101 and rebuild the patentability threshold. Replacing the old *State Street* test, CAFC devised the so-called “Machine or Transformation” (MOT) test: a claimed subject matter is eligible under Section 101 if “(1) it is tied to a particular machine or apparatus, or (2) it transforms a particular article into a different state or thing.”¹²³ Unfortunately, CAFC’s MOT test did not go down well with judges at the Supreme Court, which nevertheless also ruled *Bilski*’s claims unpatentable by applying a different test later in 2010. In the Supreme Court’s

¹²⁰ 149 F. 3d 1368 at 1373 (Fed. Cir. 1998)

¹²¹ Mark Lemley, Michael Risch, Ted Sichelman, R. Polk Wagner, “Life after *Bilski*” (2011) *Stanford Law Review* 101 at 103

¹²² It is observed that there was immediately a surge of “software patents” after the *State Street* ruling. See Kretschmer, Section 2.5, *supra* note 39

¹²³ 545 F.3d 943, at 954 (Fed. Cir. 2008)

Bilski decision, the majority opinion by Justice Kennedy points out that the MOT is a flawed test and it should not be relied upon as the sole test for deciding patent eligibility but it is merely “a useful and important clue.”¹²⁴ Also based on a review of three earlier cases *Benson*, *Flook and Diehr*, Kennedy argues that the correct ground for refusing *Bilski*’s claims is that the claimed subject matter is an “abstract idea”, which is a well-established excluded subject matter in US case law.¹²⁵ However, this “abstract idea” test is not uncontroversial either. Lemley *et. al.* argue that the Supreme Court’s exclusion of “abstract idea” from being a patentable subject matter is unsatisfactory, because “[n]o class of invention is inherently too abstract for patenting.”¹²⁶ The true reason has to do with the negative impact of a broad patent claim that may have on downstream innovation: “Rather, the rule against patenting abstract ideas is best understood as an effort to prevent inventors from claiming their ideas too broadly.”¹²⁷ It is also interesting to note that the MOT test, far from being reduced to merely “a useful and important clue” by the Supreme Court, continues to be frequently used by the USPTO and some district courts after *Bilski*, partly because of the practical difficulty of applying Supreme Court’s “abstract idea” test to determine patent eligibility.¹²⁸

In summary, the history from *Benson* to *Bilski* shows that the conceptual reach of “software patent” in the US context is also difficult to pin down. Although the US patent system has gone out of its most generous period of granting “software patents” ten years after the CAFC’s *State Street* decision, the Supreme Court *Bilski* decision does not really bring more certainty to the issue. In this sense, the US shares the same type of struggle with Europe, which has been unable to find one single consistently applied test of “technical character” to qualify patentable software-related “inventions”. Just as Pila observes that the US Supreme Court *Bilski* ruling only “has left the scope of US law substantially uncertain, and underlined the

¹²⁴ 130 S.Ct. 3218 at 3227 (2010)

¹²⁵ *ibid.*

¹²⁶ Lemley *et. al.*, “Life after *Bilski*”, *supra* note 121, at 102

¹²⁷ *ibid.*, at 132

¹²⁸ *ibid.*

difficulties of attempting to interpret the EPC in a manner that tracks US jurisprudence.”¹²⁹

3.4.2 Perceived Threat of Patents to Software Innovation

The above sub-section shows that although computer programs *per se* are not patentable, patents of software-related inventions have been in existence in both US and Europe since the 1980s. So how do FOSS programmers react to these patents? To which extent are patents perceived to be a threat to software freedom? Is it possible for the perceived threat to be contained? Again it would be wrong to make a sweeping statement that all FOSS developers are categorically against patents as a threat. In order to appreciate the complexity of this issue, I identify two historical stages to account for FOSS programmers’ evolving reaction to the growing influence of patents on their community. The first stage covers roughly the first decade after 1981 when the US Supreme Court issued *Diehr* decision. During this period of time, FOSS programmers were not very much aware of the actual existence of software-related invention patents, let alone their possible impact on decentralised collaborative FOSS projects. This situation is very unlike the advent of software copyright, whose impact was immediately felt and heatedly debated in the 1980s.¹³⁰ Stallman recalls that when the *Diehr* decision came out in 1981, this milestone in the history of software-related invention patents simply passed unnoticed by most programmers: “When the US started having software patents, there was no political debate. In fact, nobody noticed. The software field, for the most part, didn’t even notice.”¹³¹ This lack of awareness in part explains why early versions of GNU copyleft licences (i.e., Emacs GNU Public License in 1985 and GNU GPL v1.0 in

¹²⁹ Justine Pila, “Software Patents, Separation of Powers, and Failed Syllogisms: A Cornucopia from the Enlarged Board of Appeal of the European Patent Office, (2010) *Oxford Legal Research Paper Series*, Paper No 48/2010, p.7

¹³⁰ Recall that Stallman’s knee-jerk reaction to copyright was that it was “blasphemous” to hackers’ world in the early 1980s. See Section 2.3.1, Chapter 2 and Steven Levy, *Hackers*, p.419

¹³¹ Stallman, “Software Patents—Obstacles to Software Development”, script of a speech delivered at the University of Cambridge Computer Lab, 25 March 2002 at <<http://www.cl.cam.ac.uk/~mgk25/stallman-patents.html>>

1989) are merely *copyright* licences that did not mention the threat of patents to free software at all.¹³² This situation would change later in the 1990s.

The problem of software patents to software freedom was largely hidden in the 1980s, but it gradually surfaced when it reached its second stage since the early 1990s. In June 1991, when Stallman upgraded GNU GPL to Version 2.0, he added a preambular text alerting that “any free program is threatened constantly by software patents”:

We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.¹³³

Around the same period of time, there has also been no shortage of anecdotal stories vindicating GPL's warning of the disruptive effect of patents to software projects. For example, in September 1991, Stallman himself was forced to abandon a data compression program contributed by a volunteer programmer. This is because, just about one week before a release of GNU software, Stallman accidentally found a newly issued patent that *might* “read on” this contributed compression program.¹³⁴ Note that the risk of this patent as assessed by Stallman is merely potential but not actual. Lemley and Shapiro point out that a patent is not an absolute right to exclude but merely a “probabilistic” one if it is not litigated in court.¹³⁵ It is not rare that risk-averse FOSS project leaders like Stallman would choose to be on the safe side by declining contributions that are *probable* to infringe, because these volunteer-driven FOSS projects cannot afford to be bogged down by a hugely costly litigation in the first place, even though the patent involved might be proved legally invalid when litigated in court.

¹³² In the 1980s there was hardly any patent dispute that affected the FOSS community, though software copyright disputes were already not uncommon among programmers. The dispute between Stallman and Gosling over Emacs copyright is an obvious example. See Chapter 2 for detail.

It was not until 2007 that GPL 3.0 for the first time introduced a clause specific to patent issues.

¹³³ Preamble, GPL v2.0

¹³⁴ See Stallman, “Patent Reform Is Not Enough” at <<http://www.gnu.org/philosophy/patent-reform-is-not-enough.html>>

¹³⁵ Mark Lemley and Carl Shapiro, “Probabilistic Patents”, (2005)19 (2) *Journal of Economic Perspectives* 75

Although anecdotal evidence like the above abounded, a comprehensive and systematic assessment of the threat of patents was not available until the League for Programming Freedom published a report entitled *Software Patents: An Industry at Risk* in 1994. The title of this report is self-explanatory enough to show LPF's anti-patent position, which openly advocates to "abolish software patents completely" and that "software be made explicitly non-patentable."¹³⁶ More specifically, the report identifies six important reasons to support their patent-abolitionist policy. First, software is a highly complicated and sophisticated artifact which aggregates a huge number of technical components. Any of these components may unintentionally infringe upon patented technologies, which "make the legal risks and expenses associated with developing even well understood software frightening."¹³⁷ Secondly, apart from being a complex artifact, "the nature of software means that much of it is also very abstract". The highly abstract nature makes many complex components of software unable to be neatly separated and analysed. This abstractness of software will lead to the abstractness of "software patents" that may potentially give its owner a broad range of monopoly. In short, "software's abstraction makes it difficult to partition these technologies" and exactly for this reason software patents are very expensive to search, analyse and litigate in court.¹³⁸ Retrospectively, LPF's second argument, that software is too "abstract" to be patentable to some extent is not qualitatively different from the US Supreme Court's 2010 *Bilski* majority opinion that a claimed subject matter cannot be patentable if it is an "abstract idea".¹³⁹ Thirdly, the duration of patents is too long to suit software technologies that grow at a rapid rate. "This rapid rate of evolution means that those who are investing time creating and lodging patents are vastly outpacing those who are investing effort bringing such ideas to market. By the time an immature technology develops to the point where it can be incorporated into products, it has a dozen or more patents on it that render it commercially intractable."¹⁴⁰ Fourthly, software is not like other physical consumer goods that can wear out. "A computer program that is fully debugged will perform its function forever without requiring maintenance or

¹³⁶ Original texts are in all in capital letter. Gordon Irlam and Ross William, *Software Patents: An Industry at Risk*, 1994, at <<http://www.progfree.org/Patents/industry-at-risk.html>>

¹³⁷ *ibid.*, Section 2.1

¹³⁸ *ibid.*, Section 2.2

¹³⁹ 130 S. Ct. 3218 (2010)

¹⁴⁰ *ibid.*, Section 2.3

modification”. So, in order not to lose customers, software companies have to keep updating and adding new features to their products and “the industry will remain innovative whether or not software patents exists”.¹⁴¹ Fifthly, software patents add huge legal costs to software development and would eat into the resources that should have gone into software innovation itself. Very often patents are employed as a strategic weapon to lock out competitors. Especially for those individual programmers and small companies who lack a legal infrastructure to defend themselves, “the prospect of being sued over a patent infringement even if the case is ungrounded and would ultimately fail is so terrifying, that many companies choose to give all patents they know about a wide berth rather than risk the possibility of any kind of patent challenge.”¹⁴² Finally, software’s commercial success relies on their “market-driven properties” more than their being given a monopoly protection for being absolutely “novel” as defined by the patent system.¹⁴³ Software companies become market leaders not because they are the very first to invent a particular “new” technology, but because they are more attentive and adaptive to the consumers’ need for high-quality software products. In summary, the six arguments above show that the complex and abstract nature make software ill-suited to the patent system, and thus a “vision of patents entrenched in the software industry is a vision of stagnation.”¹⁴⁴

The LPF Report has presented an abolitionist argument *tour de force* by painting a dark and gloomy prospect of the software industry being plagued by the patent system.¹⁴⁵ However, this prospect may well be exaggerated according to some academic commentators. For example, with the benefit of hindsight, software-related

¹⁴¹ *ibid.*, Section 2.4

¹⁴² *ibid.*, Section 2.5

¹⁴³ The report’s examples are illustrative of the point: “Borland didn’t invent compilers. Microsoft didn’t invent operating systems. Novell didn’t invent networking. Sun didn’t invent Unix. Apple didn’t invent the graphical user interface. Oracle didn’t invent the database. It turns out that nearly all successful software companies have concentrated on constructing better implementations of already existing technologies.” *ibid.*, Section 2.5

¹⁴⁴ *ibid.*

¹⁴⁵ It is important to note that what is meant by “software industry” in the LPF Report includes both FOSS and proprietary software sectors, to which “software patents” are arguably a threat. In fact, the LPF Report does propose some options to reform the US patent system, though these options are not favoured over total abolition. The proposed changes include: “1) Tighten up the requirements for awarding software patents. 2) Reduce the duration of software patents from 17 years to, say, 3 years. 3) Significantly reduce the period of pendency. 4) Find a simpler way to determine if a piece of code is affected by a patent. 5) Improve patent indexing so that software patents can be more easily searched. 6) Publish patent applications as soon as they are received.” See Section 5.1, *ibid.*

invention patents, as Merges observes, have not been able to entirely stifle innovation in the software industry after 1994. Patents have posed certain a risk to software innovation, but this risk is by no means a devastating one. Merges argues that the LPF, among other early patent abolitionists, is mostly wrong: “Patents have not killed the software industry, they have not led to a slowdown in entry, and they do not appear to have assisted in the entrenchment of large companies at the expense of smaller and newer ones. Despite the predictions of the League for Programming Freedom, the industry has not stagnated.”¹⁴⁶

Not all those who participate in, or sympathise with, the FOSS movement, share LPF’s abolitionist position. Some of them are less keen to abolish than to reform the patent system. The emergence of patent reformism within the FOSS community roughly coincides with the spin-off of the pragmatist “open source” approach from the purist “free software” approach. The “open source” campaign has made non-proprietary software programming friendlier and more attractive to commercial software companies, many of which can afford to defend themselves against patent infringement allegations, or even build their own defensive patent portfolios. This has given rise to an interesting phenomenon of “open source patents” named by Leveque and Ménière, to account for those patents owned by corporate open source developers.¹⁴⁷ For example, IBM is probably the most well-known “open source patents” owners. In 2005, IBM decided to “donate” 500 patents to the FOSS community. This “donation” was in the form of a pledge not to assert the 500 named patents against any FOSS project under a licence approved by the Open Source Initiative as of 1 November 2005.¹⁴⁸ It is worth noting that these 500 patents only form a very small part of IBM’s whole patent portfolio. IBM’s pledge is not an ideological commitment to the “free software” ideal but largely a strategic move. Haas points out that profits can still be extracted from IBM’s non-pledged patents, which simply become more important assets to the company: “IBM may actually be giving up very little in its pledges, since the patents in the pledge may or may not

¹⁴⁶ See Robert P. Merges, “Software and Patent Scope: A Report from the Middle Innings” (2007) 85 *Texas Law Review* 1627 at 1632

¹⁴⁷ Francois Leveque and Yann Ménière, “Copyright Versus Patents: The Open Source Software Legal Battle” (2007) 4(1) *Review of Economic Research on Copyright Issues* 27 at 42

¹⁴⁸ See IBM, “IBM Statement of Non-Assertion of Name Patents against OSS” at <<http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf>>

have value as revenue generators. IBM does not provide non-assertion guarantees for its ostensibly profitable closed source products or patent holdings.”¹⁴⁹ It is important to know that IBM is not the only company that has “open source patents”. Another interesting example is the Open Invention Network (OIN), which is a consortium initially formed by five companies, including Red Hat, IBM, Sony, Novell and Philips in November 2005. It has acquired hundreds of “open source patents”, which are then made “available royalty-free to any company, institution or individual that agrees not to assert its patents against the Linux System” under an OIN licence, which is not hugely dissimilar from agreements used by conventional cross-licensing patent pools.¹⁵⁰ In short, corporate FOSS developers perceive the threat of software invention patents differently from non-corporate volunteer FOSS programmers. The former believe that the abolition of patents is unnecessary largely because they have the resources to defend themselves, whilst the latter perceive patent infringement allegations are devastating to software freedom within community-led projects. I will show in *infra* Section 3.5.2 how Stallman, from the non-corporate FOSS developers’ perspective, insists on patent abolitionism and at the same time uses the GPL to minimise the patents’ threat to software freedom.

3.5 GPL and Software Freedom

Based on the legal background concerning copyright and patent as introduced by previous sections, this section further examines the first and most prominent FOSS licence—GNU Public Licence (GPL)—and its struggle to find an accurate legal expression of software freedom since the mid-1980s. It shows that the drafters of the GPL are attentive to the subtle differences between copyright and patent, which will be discussed separately below.

3.5.1 GPL as a Copyright and “Copyleft” Licence

It has been shown in Chapter 2 that the GPL came out of a unique period when the old hackers’ stewardship duty to preserve software commons clashed intensely with

¹⁴⁹ See Douglas A. Haas, “A Gentlemen’s Agreement—Assessing the GNU General Public License and Its Adaptation to Linux” (2007) 6 *Chicago-Kent Journal of Intellectual Property* 213 at 276

¹⁵⁰ For a list of OIN-owned patents, see OIN, “Open Invention Network’s Currently Owned Patents”, last accessed 28 May 2011, at <http://www.openinventionnetwork.com/pat_owned.php>

the rising proprietary right to own software privately allowed by copyright in the 1980s. From 1983 to 1985, Richard Stallman was embroiled in a copyright dispute with James Gosling over a version of the Emacs programming editor, which was initially developed collaboratively by computer hackers since the 1970s. Gosling's decision to withdraw and privatise his contribution caused great tension in the Emacs community. During this dispute, Stallman gradually familiarised himself with US software copyright law, which eventually led him to produce the Emacs GPL in 1985.¹⁵¹ This Emacs-specific GPL, which is the predecessor of three later versions of the general-purpose GNU GPL, makes it clear that no Emacs user should be deprived of the rights to freely use, copy, change and redistribute the program in any future distribution:

Specifically, we [i.e., Emacs programmers] want to make sure that you [i.e., users] have the right to give away copies of Emacs, that you receive source code or else can get it if you want it, that you can change Emacs or use pieces of it in new free programs, and that you know you can do these things. To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of Emacs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.¹⁵²

The actual terms and conditions of the Emacs GPL licence are specified in the five “Copying Policies” drafted by Stallman. Among these five, the most important one is no doubt the second “Copying Policy” that innovatively devises a “copyleft” provision, obligating downstream programmers to share their publicly released contributions of Emacs under the same licence: “You may modify your copy or copies of GNU Emacs source code or any portion of it, and copy and distribute such modifications [...], provided that you [...] cause the whole of any work that you distribute or publish, that in whole or in part contains or is a derivative of GNU Emacs or any part thereof, to be licensed at no charge to all third parties on terms identical to those contained in this License Agreement [...]”¹⁵³ This is the situation

¹⁵¹ For the history of the Emacs dispute, see Section 2.3.2 in Chapter 2 for more detail.

¹⁵² See Emacs GPL (first published in 1985 and later clarified 11 Feb 1988)

¹⁵³ *ibid.*

where a “copyleft” clause is used for the first time in the software licensing history and it remains a defining feature of all later GPL licences.

It is not difficult to find that the invention of “copyleft” in the Emacs GPL is to mimic the pre-copyright environment where software is not exclusively owned by any single programmer but it is collaboratively created and preserved in a commons.¹⁵⁴ Paradoxically, copyleft’s attempt to secure software freedom is mainly couched in the legal language of copyright, which has given a wide scope of exclusive rights to software authors. This paradox of using a copyright licence to create a non-exclusive property regime is pointed out by Steven Weber: “property in open source is configured fundamentally around the right to distribute, not the right to exclude.”¹⁵⁵ The 1985 Emacs GPL is no doubt a first step of an intrepid long journey of experimenting with a licensed non-exclusive software commons. Although this licence later has been replaced by the three generations of the general-purpose GNU GPL respectively published in 1989, 1991, 2007, the initial design of a copylefted commons has remained largely unchanged. I will use the latest GPL 3.0 as an example to show how the initial legal scaffolding is preserved more than two decades after the first Emacs GPL was created.

Although GPL 3.0 is a much longer and more detailed document than the original Emacs GPL, the former does not deviate wildly from the latter when dealing with software copyright. There is a common licensing structure that can be broken down into three basic licensing components respectively dealing with 1) permissions, 2) conditions and 3) termination. The first component concerns clauses that give “permissions” to use, copy, modify and redistribute software in line with the Free Software Definition.¹⁵⁶ Without these permissions, these acts would otherwise be restricted by copyright law. It is worth noting that the root meaning of “licence” begins merely as “permission”: Just as Laddie *et. al.* points out “[i]n the strict sense a licence is a mere permission to do that which would otherwise be unlawful and it

¹⁵⁴ Note that Benkler’s peer-production model is exactly built in a non-proprietary environment: “individuals produce on a non-proprietary basis and contribute their product to a knowledge ‘commons’ that no one is understood as ‘owning,’ and that anyone can, indeed is required by professional norms to, take and extend.” Yochai Benkler, “Coase’s Penguin, or, Linux and ‘The Nature of the Firm’” (2002) 112, (3) *Yale Law Journal* 369 at 381-2

¹⁵⁵ Steven Weber, *Success*, p.1

¹⁵⁶ Richard Stallman, “The Free Software Definition” at <<http://www.gnu.org/philosophy/free-sw.html>>

confers no proprietary rights on the licensee.”¹⁵⁷ Along the same line, Bently and Sherman point out:

At a basic level a licence is merely a permission to do an act that would otherwise be prohibited without the consent of the proprietor of the copyright. A licence enables the licensee to use the work without infringing. So long as the use falls within the terms of the licence, it gives the licensee an immunity from action by the copyright owner.¹⁵⁸

The GPL 3.0 makes it clear that it is a copyright licence that gives permission: “nothing other than this License grants you *permission* to propagate or modify any covered work. These actions infringe copyright if you do not accept this License”¹⁵⁹ (added emphasis). Apart from giving normal copyright permission, it is interesting to note that GPL 3.0 makes a new special permission that does not exist in earlier versions in the GPL family. It permits users to circumvent Digital Right Management (DRM) technologies if DRM is used in GPL covered works. This permission is drafted in response to the rise of the anti-circumvention law introduced by 1996 WIPO Copyright Treaty (WCT) that forbids “the circumvention of effective technological measures that are used by authors in connection with the exercise to their rights under this Treaty or the Berne Convention and that restrict acts, in respect of their works, which are not authorised by the authors concerned or permitted by law.”¹⁶⁰ This WCT anti-circumvention clause and its progeny¹⁶¹ are sometimes known as “para-copyright”, because the protected technological measures are not copyright measures themselves but they have the effect of expanding the scope of authors’ exclusive rights and potentially upsetting the balance intended by copyright law.¹⁶² The drafter of the GPL 3.0 believes that DRM is “fundamentally in conflict

¹⁵⁷ It is also pointed out that there is no reason why a licence should not be given “to the world at large or to a specified section of community”. See Hugh Laddie, *et al.*, *Modern Law of Copyright and Designs*, p. 903-4;

¹⁵⁸ Although a licence begins with “merely a permission”, the copyright system has nurtured “a sophisticated repertoire of ways whereby a work might be licensed”. Lionel Bently and Brad Sherman, *Intellectual Property Law* (Oxford: OUP, 2009, 3rd Ed.) p.264

¹⁵⁹ Section 9, GPL 3.0

¹⁶⁰ Art. 11, WCT

¹⁶¹ Equivalents of Art. 11, WCT can be found in the US Digital Millennium Copyright Act, 17 U.S.C 1201 and Art. 6, European Copyright Directive.

¹⁶² For a critique of the anti-circumvention law, see, for example, Dan Burk, “Anticircumvention Misuse” (2003) 50 *UCLA Law Review* 1095

with the freedoms of users that the GPL is designed to safeguard”.¹⁶³ So in order to counter this, GPL 3.0 states that “[n]o covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.” As a consequence, programmers of GPL covered software waive their anti-circumvention right:

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.¹⁶⁴

In short, under the above anti-anti-circumvention clause, GPL 3.0 effectively extends permission to acts that would otherwise be restricted by the para-copyright created by technological measures.

The second component of the GPL is its “conditions”. GPL is often known as a “conditional licence”, because it sets up a series of carefully crafted “conditions” to safeguard software freedom, under which users are permitted to use, copy, modify and distribute GPL covered works. Largely inheriting a structure used by the Emacs GPL, GPL 3.0 divides itself into three categories—including “*conveying* verbatim copies”, “*conveying* modified source versions” and “*conveying* non-source forms”—all of which will trigger the “conditions” attached to the “permissions”. (See Table 3.1)

It is important to know that not all acts of running, copying or modification of GPL covered software will trigger the conditions stated in GPL. Doing these acts *privately* is permitted unconditionally. Conditions apply only when “verbatim copies” or “modified source versions” or “non-source forms” are *conveyed to the public*. Note that GPL 3.0 deliberately avoids the familiar term “distribute”, which is used in US

¹⁶³ FSF, *GPLv3 First Discussion Draft Rationale*, 16 January 2006, Section 2.4, at <<http://gplv3.fsf.org/gpl-rationale-2006-01-16.html>>

¹⁶⁴ Section 3, GPL 3.0

copyright law¹⁶⁵ and all previous versions of GPL, but it adopts two unfamiliar terms “conveying” and “propagation”. The reason behind this is that GPL 3.0 is intended to be globally applicable and the term “distribution” in different jurisdictions may have different meanings. The drafter of GPL 3.0 explains: “The scope of ‘distribution’ in the copyright context can differ from country to country. We do not wish to force on the GPL the specific meaning of ‘distribution’ that exists under United States copyright law or any other country’s copyright law.”¹⁶⁶ For example, in a non-US jurisdiction such as UK, “distribution” does not cover the copyright holders’ exclusive right to “communicate the work to the public”¹⁶⁷ which is not explicitly mentioned in the US statutory language. This right was harmonised by the EU Copyright Directive, which provides that authors must have, as part of their right to control public communication, the “exclusive right to authorise or prohibit any communication to the public of their works in such a way that members of the public may access them from a place and at a time individually chosen by them.”¹⁶⁸ This right is tailored to Internet transmissions such as the making available of copyright materials to the public (for example, via a P2P file-sharing network).¹⁶⁹ Clearly, the GPL would be failing in its purpose if it did not cover transmissions of this nature. In this light, Stallman, as the main drafter of GPL 3.0, chose the term “propagation”, which is not used by any particular legal system.¹⁷⁰ Here is a definition of “propagation” offered by GPL 3.0:

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under

¹⁶⁵ 17 USC 106

¹⁶⁶ FSF, *GPLv3 First Discussion Draft Rationale*, supra note 163, p.8

¹⁶⁷ s.20 CDPA

¹⁶⁸ Art. 3(1) Directive 2001/29/ EC of the European Parliament and of the Council of 22 May 2001 on the Harmonisation of Certain Aspects of Copyright and Related Rights in the Information Society (EU Information Society Directive)

¹⁶⁹ Bainbridge observes: “A great deal of software, including computer programs and associated works, is now available for downloading online. Much is free to download but, of course, still protected by copyright and further copying and distribution over and above that allowed by the licence agreement, or other terms under which it is made available, will infringe copyright. Some of the software available on websites of dubious pedigree may be infringing software.” Bainbridge, supra note 23, p.82

¹⁷⁰ Propagation “is a term not tied to any statutory language. Propagation that does not enable other parties to make or receives copies—for example, making private copies or privately viewing the program—is permitted unconditionally, propagation that does not enable other parties to make or receive copies is permitted ‘distribution,’[...]” FSF, *GPLv3 First Discussion Draft Rationale*, supra note 163, p.11

applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.¹⁷¹

Not all acts of “propagation” will trigger the conditions in GPL 3.0, but only a subset of it known as “conveying” will: “To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.”¹⁷² To put it another way, a user can *privately* “propagate” the GPL covered software unconditionally, but “conditions” will apply only when copies of the software are “conveyed” *to other parties*. The second paragraph in Section 3, GPL 3.0 clarifies this point: “You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force.”

There is no doubt that the most famous and important “condition” in GPL is its “copyleft” requirement. Most basically, copyleft mandates using the same licence when *conveying to the public* a “modified source version” of the original GPL covered software. Section 5(c) stipulates:

You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply [...] to the whole of the work, and all its parts, regardless of how they are packaged.¹⁷³

This above quoted copyleft requirement is also known as the “viral” clause, because it seems able to “contaminate” any work that has established some level of connection with GPLed code.¹⁷⁴ However, the virality of copyleft is sometimes

¹⁷¹ Section 0, “Definition”, GPL 3.0

¹⁷² Para. 7, Section 2, GPL 3.0

¹⁷³ Section 5 (c), GPL 3.0

¹⁷⁴ However, it has never been clear about the exact extent to which a GPL program is closely connected enough to “contaminate” a non-GPL program. The issue has never been tested in court. Raymond and Raymond find there are at least four competing theories on the issue. See Eric Raymond and Catherine Olanich Raymond, *Licensing HOWTO*, 9 November 2002, at <<http://catb.org/~esr/Licensing-HOWTO.html>>

Also it seems very difficult to come up with a general theory about the issue, which is always quite project-specific. Lead programmers may have their own interpretation according to the nature of their

unduly exaggerated.¹⁷⁵ It is often overlooked that there are at least three limiting factors that would circumscribe the reach of copyleft. First, GPL does not “contaminate” privately made modifications. So long as a modification is not publicly conveyed, no condition applies. Secondly, GPL does not contaminate so-called “compilations” in which a GPL covered work and other programs with which it is aggregated are separate and independent from each other, even though they are stored on the same distribution medium. So long as these programs are not combined into one larger integrated program, the “compilation” remains an “aggregate”, which is outside the reach of copyleft.¹⁷⁶ Thirdly, GPL does not seem to be able to “contaminate” a program that only *unwittingly* incorporates GPL covered code. For example, Epstein conceives a “nightmare scenario” where a Microsoft employee-programmer incorporates a piece of GPLed program into Microsoft’s proprietary operating system without the company’s knowledge.¹⁷⁷ So should Microsoft worry that its whole operating system is now irreversibly “contaminated” and thus fall under the reach of GPL? Kumar argues that this worry is unfounded. Because the company has no knowledge of the licence and it does not really “accept” the condition and thus “there is no meeting of the minds”. In other words, the copyleft provision is not contractually binding on the company.¹⁷⁸ However, this view is not uncontroversial. The FSF’s official jurisprudence is that the GPL is not a contractual licence but a pure copyright licence. So the validity of copyleft does not depend on

own projects. For example, Linus Torvalds argues that the Linux kernel does not contaminate the user programs that run on it: “This copyright does not cover user programs that use kernel services by normal system calls—this is merely considered normal use of the kernel, and does not fall under the heading ‘derived works.’” quoted in Robert W. Gomulkiewicz, “A First Look at General Public License 3.0”, (2007) 24 *Computer and Internet Lawyer* 15 at 15

¹⁷⁵ For example, there has been no shortage of smear campaigns by some proprietary companies to exaggerate the danger of the viral nature of copyleft. See Andres Guadamuz, “Viral Contracts or Unenforceable Documents?” *Contractual Validity of Copyleft Licences* (2004) 26 (8) *EIPR* 331; Guadamuz, “Legal Challenges to Open Source Licences” (2005) 2 (2) *SCRIPT-ed* 163

¹⁷⁶ The last paragraph in Section 5, GPL 3.0 makes this point clear: “A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.”

¹⁷⁷ Richard Epstein, “Why Open Source is Unsustainable”, *Financial Times*, 21 Oct 2004 at <<http://www.ft.com/cms/s/2/78d9812a-2386-11d9-ae5-00000e2511c8.html>>

¹⁷⁸ Sapna Kumar, “Enforcing the GNU GPL” 2006 *University of Illinois Journal of Law, Technology and Policy* 1 at 18

whether there is a valid contract or not.¹⁷⁹ To fully assess the legal validity of GPL is beyond the purpose of this chapter but it will be scrutinised in detail in Chapter 4.

The third component of the GPL is its clause on when and how to “terminate” the licence when the second component, i.e., “condition”, is breached. It is interesting to note that the termination clause in GPL is based on software authors’ property “right to exclude” and it has a role to play in securing software freedom. McGowan famously argues that “[o]pen-source production rests ultimately on the right to exclude” on the ground that this exclusionary right can be employed to discipline or deter violation of FOSS licences.¹⁸⁰ Stallman has long been aware of this disciplinary and deterrent function and he did write a termination clause into the Emacs GPL in 1985:

You may not copy, sub license, distribute or transfer GNU Emacs except as expressly provided under this License Agreement. Any attempt otherwise to copy, sub license, distribute or transfer GNU Emacs is void and your rights to use GNU Emacs under this License agreement shall be automatically terminated.¹⁸¹

This text remains largely unchanged in the versions 1.0 and 2.0 of the later general-purpose GPL. GPL 3.0 amends the termination clause to make it more lenient to violators of the licence. It allows grace time for violators to cure the violation themselves and then provisionally or permanently reinstate the licence.¹⁸² This change is intended to alleviate the harshness of the automatic termination of the licence and incentivise violators to correct their own mistakes as soon as possible.

¹⁷⁹ “You are not required to accept this License in order to receive or run a copy of the Program.” Section 9, GPL 3.0

¹⁸⁰ See David McGowan, “Legal Impactions of Open-Source Software” (2001) *University Illinois Law Review* 241 at 303; However, Benkler does not agree with McGowan’s view. He suggests that there can be alternative institutional arrangements to replace the exclusionary property system: “The same protection from defection might be provided by other means as well, such as creating simple public mechanisms for contributing one’s work in a way that makes it unsusceptible to downstream appropriation—a conservancy of sorts.” Benkler, “Coase’s Penguin”, supra note 154 at 446

¹⁸¹ Copying Policy 4, Emacs GPL, 1985

¹⁸² Paras. 2&3, Section 8, GPL 3.0

Table 3.1 Copyright: Conveying GPL Covered Works					
	Emacs GPL (1985)	GPL1.0 (1989)	GPL2.0 (1991)	GPL3.0 (2007)	
Conveying Verbatim Copies (<i>Permission & Condition</i>)	Copying Policy 1	Section 1	Section 1	Section 4	
Conveying Modified Source Versions (<i>Permission & Condition</i>)	Copying Policy 2	Section 2	Section 2	Section 5	
	Copyleft	Section 2 (b)	Section 2 (b)	Section 5 (c)	
Conveying Non-Source Forms (<i>Permission & Condition</i>)	Copying Policy 3	Section 3	Section 3	Section 6	
Termination (<i>Violation of Condition</i>)	Copying Policy 4	Section 4	Section 4	Section 8	

3.5.2 GPL as a Patent Licence and its Limit

In the first decade after the 1981 *Diehr* decision, software-related invention patents were not immediately perceived as a palpable threat to software freedom.¹⁸³ It is not surprising that 1985 Emacs GPL and 1989 GNU GPL 1.0 did not mention patents at all. However, awareness of the negative impact of patents on software freedom was gradually built up from the early 1990s. In 1991, the text of GPL 2.0 for the first time condemned “software patents” as a threat, but it still did not give an explicit patent licence. In 2007, FSF substantially amended the licensing terms concerning patents in GPL 3.0 in order to partially contain the growing threat from patents. (See Table 3.2)

More specifically, there are four places where patents are explicitly dealt with in GPL 3.0. Firstly, the preamble reiterates FSF’s traditional anti-patent position, and it also signals that some changes have to be made in GPL 3.0: “[...] 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.”¹⁸⁴

¹⁸³ See above Section 3.4.2 of this chapter.

¹⁸⁴ Para. 9, Preamble, GPL 3.0

Secondly, Section 11 is a newly added clause explicitly granting a patent licence from GPL software contributors to users: “Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.” It is noteworthy that a patent licence like this is not the invention of GPL 3.0, but the FSF has largely borrowed the idea from the Apache License, which is a pioneer in dealing with patents in relation to FOSS contribution.¹⁸⁵

Thirdly, a licensee should not initiate patent litigation in respect of a GPL covered work. The consequence of asserting patent rights will trigger the termination clause in Section 8, which would stop the patentee-licensee from using the licensed work any further. This is because the condition in Section 10 stipulates that a licensee “may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it”. The combination of Sections 8 and 10 effectively functions as a patent-retaliation clause, which has already been used by some other FOSS licences before.¹⁸⁶

Finally, GPL 3.0 contains a so-called “liberty-or-death” clause: a programmer should not convey a piece of code to a GPL project, if he is encumbered with an external obligation (for example, to collect a patent royalty) that is in contradiction with the conditions of the GPL. Section 12 reads: “If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all.” Note that this

¹⁸⁵ The patent licence granted by Apache reads: “Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted.” Section 3, Apache Licence 2.0

¹⁸⁶ To use Apache as an example again, the retaliation clause of the Apache Licence says: “If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.” *ibid.*

clause in GPL 3.0 is not entirely new either, but it is a slight revision of Section 7 of GPL 2.0.

The above four amended parts do not intend to fully, but only partially, contain the threat from patents. There is a limit to what a licence can achieve. Stallman believes that the root of the whole problem lies in the legal system issuing “software patents” in the first place.¹⁸⁷ The changes made in GPL 3.0 only reflect a pragmatic move to work *with* rather *against* the existing patent regime in order to minimise the threat from patents to software freedom. GPL, however more legally sophisticated it may be, is simply not able to eliminate the root problem of “software patents” altogether.

	Emacs GPL (1985)	GPL1.0 (1989)	GPL2.0 (1991)	GPL 3.0 (2007)
Preambular text condemning “software patents”	N/A	N/A	Preamble	Preamble
Liberty-or-Death Clause	N/A	N/A	Section 7	Section 12
Patent Licence	N/A	N/A	Implicit	Section 11
Patent Retaliation	N/A	N/A	N/A	Sections 8&10

3.6 Conclusion

This chapter is a survey of two types of “intellectual property”—copyright and patent—and their respective impact on software freedom. FOSS programmers do not endorse “intellectual property” as a unified body of law, but argue that a more subtle understanding is required. There are three observations coming out of a scrutiny of the subtleties of this issue. Firstly, FOSS programmers generally endorse copyright, which is the main legal basis for them to license their software in a non-exclusive fashion. However, they are against some companies’ efforts to stretch copyright further to cover the non-expressive part of software. Secondly, standalone software is normally not a patentable subject matter in either EPC countries or the US. Under the EPC jurisprudence, the patentability of a software-related invention depends on whether the claimed subject matter has the right kind of “technical character”. Since the 1987 *Vicom* decision until now, the EPO has failed to apply a single consistently used interpretation about the meaning of “technical character”, a fact that has led to

¹⁸⁷ Stallman, Richard, “Fighting Software Patents—Singly and Together”, 2004, at <<http://www.gnu.org/philosophy/fighting-software-patents.html>>

great uncertainty over this issue. In the US, from the 1972 *Benson* case to 2010 *Bilski* case (via the 1998 *State Street* case), its legal system has also been struggling to produce a suitable test to assess the patentability of software-related inventions. Among FOSS programmers, patents have remained a divisive issue. Non-corporate volunteer FOSS programmers tend to have an anti-patent position because they are more vulnerable to patent infringement allegations, while corporate open source participants tend to be more interested in reforming the patent system than abolishing it altogether. Thirdly, the gradual maturing of GPL from 1985 to 2007 reflects a continuous struggle to find a way to protect software freedom in an ever-changing legal climate. GPL is primarily a copyright licence that creates a unique copyleft mechanism to build an unbroken chain of software freedom. It also responds to programmers' growing concern about patents by substantially amending its latest version of GPL in order to partially contain the threat from patents. The next chapter will discuss the legal validity of the GPL and some other licences.

Chapter 4 Understanding FOSS Licences as Standard Forms—A Relational Contract Perspective

4.1 Introduction

As has been discussed in previous chapters, the main goal of FOSS licensing is to securing software freedom in radically decentralised FOSS projects. As programmers may well have different and evolving expectations about the outcomes of their collaborative efforts, it is important for a licence to standardise the minimum legal commitments for all contributors in order to prevent a given FOSS project from freewheeling into a Babel of legally incompatible fragments. These legal commitments, when verbalised by the licences, must be pursuant to the FOSS stewardship responsibility under the Free Software Definition and the Open Source Definition.

Although the goal that FOSS licences intend to achieve is undoubtedly important, the legal basis on which these licences are made enforceable is not always clear. This is largely due to the fact that FOSS licences are mainly take-it-or-leave-it standard forms, which are electronically disseminated alongside software through the internet on a mass scale. These licences do not seek affirmative assents from licensees or adhering parties through traditional bargained-for exchanges, but they are most likely to be given in either of the two types of electronic standard forms, i.e. clickwrap and browsewrap. The clickwrap requires users to click through the “Yes, I Agree” button before downloading or installing a particular piece of software, while the browsewrap is merely an electronic notice containing licensing terms and conditions. As most users do not read, let alone fully digest, all information contained in clickwrap or browsewrap licences, their assents are said to be “presumed” rather than “actual”.¹ The upshot is that there seems to be no obvious moment when the meeting of minds between licensors and licensees unequivocally happens in a non-bargained-for process like this. When put under the strict scrutiny of classical contract law, the

¹ Nancy Kim, “Clicking and Cringing”, (2007) 86 *Oregon Law Review* 797; Nancy Kim, “The Software Licensing Dilemma” (2008) *Brigham Young University Law Review* 1103

absence of “actual” assents or that of the meeting of minds poses a serious challenge to the legal validity of all sorts of non-negotiated standard forms², and FOSS licences are no exception.

In this chapter, I attempt to show that standard form FOSS licences are better understood through the lens of Relational Contract Theory (RCT) than they are through the lens of the classical contract model. There are two equally important reasons for software licensing jurisprudence to incorporate insights from the RCT. First, proponents of RCT believe that the total obligation does not merely arise from a single moment when parties’ minds are supposed to meet, but more realistically the obligation may also be shaped by ongoing relations among parties. In FOSS projects, contributors’ consent to their obligation of making contribution takes place in a more incremental way and are often derived from rich collaborative relations among contributors. It is worth noting that RCT does not make “consent” completely irrelevant in a standard form. To the contrary, RCT only alleviates the heavy burden on explicit “consent” as the sole legitimating mechanism of imposing obligations against the adhering parties. Macneil, as the main exponent of this approach to contracts, includes “effectuation of consent” as one of the common contract norms and he believes that consent still plays an important triggering mechanism in consensual relations.³

Secondly, a sustaining FOSS project relies on rich and dynamic *collaborative relations* among contributors in a community, but it is not a product from a single or even multiple *discrete transactions* between utility-maximising strangers as understood by classical contract theory. For this reason, RCT is an appropriate theoretical tool, which helps us to imagine how the collaborative relations in FOSS could be recognised and managed. In particular, the RCT approach differentiates itself from the influential (but not uncontroversial) jurisprudence developed from the landmark Seventh Circuit case *ProCD v. Zeidenberg*, where Justice Easterbrook made a mass-market standard form licence contractually enforceable.⁴ In this chapter

² David W. Slawson, “Standard Form Contracts and Democratic Control of Lawmaking Power” (1971) 84 (3) *Harvard Law Review* 529

³ Ian Macneil, *The New Social Contract—An Inquiry into Modern Contractual Relations* (New Haven and London: Yale University Press, 1980), pp.49-50 (hereafter *NSC*)

⁴ 86 F.3d 1447 (7th Cir.1996)

I aim to show that the justification of FOSS licensing does not have to be built upon the controversial *ProCD* ruling which assumes that parties merely as utility-maximising agents. Instead, RCT provides a different ground to understand standard-form FOSS licensing, where a variety of non-utility-maximisation motivations need also to be taken seriously in understanding their aim to create software commons.

The rest of the chapter is divided into four parts. The first part (Section 4.2) briefly exposes the two contrasting perspectives of seeing software licensing as “relational contract” and “discrete transaction” respectively. It shows that the difference between the two is again rooted in the early conflict between the tradition of “stewarding” software as commons and “owning” software as private property since the inception of FOSS licensing. The second part (Section 4.3) identifies three possible doctrinal routes to enforcing a given FOSS licence via “contractual licence”, “bare licence” and “promissory estoppel”. It demonstrates that some difficulties of applying these existing doctrinal rules to address the legal validity of FOSS licensing warrants an exploration of the more suitable framework offered by RCT. The third part (Section 4.4) re-examines GPL by applying some insights from the relational approach. It shows that GPL as a relational “umbrella agreement” does not prescribe any substantive obligation concerning actual contributions from individual programmers, but it only specifies a few minimum obligations to ensure all peer-produced contributions are free software components that can be later aggregated together into one project. The fourth part (Section 4.5) concludes the chapter.

4.2 FOSS Collaboration: Discrete Transaction or Relational Contract?

A collaborative *relation* in a FOSS project is very different from the sum total of a host of one-shot discrete *transactions* of software code. Instead, the relation belongs to a continuum where peer-produced contributions are pieced together in a long timeline. In order to render the distinction clear, I need to spell out the difference between two approaches to software licensing. One is the “discretist” approach that treats software as stand-alone finished products developed by professional programmers for end-users, while the other is the relational approach that views software as an indefinitely long communicative process with no clear boundary between individual exchanges.

4.2.1 Discretist Approach: “Presentation” of Total Obligation

Truly discrete transactions are very rare in the real world. They are largely idealised situations assumed by classical contract law, which can be seen as a potent embodiment of “methodological individualism” in legal scholarship. This discretist view of contract artificially atomises social relations into isolated transactional segments, where parties are assumed to be complete strangers solely interested in maximising individual utility.⁵ The discretist view of contract can be traced back to Henry Maine, whose methodological individualist thinking has great influence in shaping the ideology of classical contract law. To put it bluntly, his famous observation about modern society’s transition from “status to contract” can be seen as no more than a movement of “from status to discrete transaction”.⁶

Although completely discrete transactions are a legal fiction, there may be instances fairly *close* to them. Macneil’s famous example of an *almost* discrete transaction is “a cash purchase of gasoline at a station on the New Jersey Turnpike by someone rarely travelling the road”.⁷ This purchase happens between complete strangers who are very unlikely to meet again and repeat the same transaction. However, situations like this example are extremely rare, and even doing grocery shopping in a supermarket does not fit into this kind of discrete transactional model. Supermarkets do want to establish some kind of relationship beyond simple one-shot transactions. For instance, they may well have schemes to make customers collect loyalty points. Some of them may even encourage customers to reuse plastic bags by giving “green” points. A truly discretist supermarket do not care about how “loyal” or “green” their customers are.

⁵ Campbell and Collins point out that classical contract law denies the “social character” of contract exchanges: “the classical law of contract reproduces the principal structural contradiction of bourgeois society—a society which has at its heart a denial of its social character.” David Campbell and Hugh Collins, “Discovering the Implicit Dimensions of Contracts”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. by David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003) p.26

⁶ For Maine’s methodological individualist view of society, see Edward Shils, “Henry Sumner Maine in the Tradition of the Analysis of Society”, in *The Victorian Achievement of Sir Henry Maine: A Centennial Reappraisal*, ed. By Alan Diamond (Cambridge: CUP, 2001) pp.144-5

⁷ Macneil, “Contracts: Adjustment of Long-Term Economic Relations Under Classical, Neoclassical, and Relational Contract”, (1978) 72 (6) *Northwestern University Law Review* 854 at 857

There are two salient characteristics of the discretist thinking. First, discrete transactions are treated as though they happen in a social vacuum and do not bond parties into long and sustaining relationships. They are deemed to happen among “total strangers” who are “brought together by chance” rather than “any common social structure”.⁸ In other words, parties in discrete exchanges are atomised individuals and each of them “would have to be completely sure of never again seeing or having anything else to do with the other”.⁹ Secondly, discrete transactions are transient and short-lived. They do not last beyond the point when they are consummated. The very brief life-span of transactions is essential to transactional discreteness: “everything must happen quickly lest the parties should develop some kind of a relation impacting on the transaction so as to deprive it of discreteness.”¹⁰

It is important to note that *too much bargain* would expand the lifespan of a transaction and thus risk the loss of discreteness in exchanges. Macneil observes that “bargaining about quantities or other aspects of the transaction can erode discreteness, as certainly does any effort to project the transaction into the future through promises.”¹¹ For this reason, discretists do not welcome lengthy and elaborate bargained-for exchanges leading to contract formation, because bargains may well blur the discreteness of transactions and at the same time raise the transaction cost. Interestingly, this discretists’ hostility to bargaining reveals a built-in paradox of classical contract model. An unrelenting pursuit of transactional discreteness would inevitably erode the importance of bargained-for exchanges, which are ostensibly at the heart of a binding classic contract. The sacrifice of “bargaining” for “discreteness” is a departure point where classical contract law starts to relax its requirement about “meeting of the minds” in contract formation. This relaxation results in the classical contract law rapidly (and somewhat imperceptibly) mutating into the “neo-classical” law of contract, which attempts to soften the classicist

⁸ Macneil further points out that the discretists prefer “only a barter of goods, since even money available to one and acceptable to the other postulates some kind of common social structure.” *ibid.*, at 856

⁹ *ibid.*

¹⁰ *ibid.*

¹¹ *ibid.*

doctrinal rigidity to some extent.¹² Throughout this chapter, I argue that the real hurdle to a relational understanding of FOSS licensing is not the unadulterated *classical* model itself but its *neoclassical* mutant. This neoclassical rationale in enforcing mass-market software licences is presumed in Easterbrook's *ProCD* ruling¹³, which will be shown to be fundamentally different from the Macneil's relational approach in sub-Section 4.2.3 in some more detail.

Turning Software Development into Discrete Products/Commodities

As has been discussed in Chapter 2, software development, under its early hacker custom, is by no means the production and circulation of many discrete products, but it is always a work in progress or an indefinitely long collaborative process. (The over two-decade long Linux kernel project in progress is a case in point here.) When the publicly available source code was allowed to be modified by the public under the hacker custom, it was relatively difficult for individual programmers to privatise their intellectual inputs into alienable end-products.

However, since the mid-1970s, some commercially minded programmers gradually figured out how to turn the non-discrete software development process into discrete end-products, which were a necessary prelude to selling software like any other “physical” commodity.¹⁴ In order to artificially create discreteness for software, they needed to go through two crucial steps. First, software developers needed to distribute only the object code of software without revealing the corresponding source code. Without seeing the hidden source code, it is made difficult for users to customise the software to their needs when necessary. The closed-source software thus loses its “extensibility”¹⁵, but at the same time it acquires quasi-physical “thingness” with a much more clearly defined boundary.

¹² Feinman observes that “[n]eoclassical method is a mix of rules and standards. This is still doctrine, by and large, but it is doctrine of a much softer sort than in classical law. See Jay M. Feinman, “Relational Contract Theory in Context”, (2000) 94 *Northwestern University Law Review* 737 at 739

¹³ 86 F.3d 1447 (7th Cir.1996)

¹⁴ Macneil points out that discretisation and commoditisation go hand in hand under classical contract law, which “transactionizes or commodifies as much as possible the subject matter of contracts”. See Macneil, *supra* note 7 at 863

¹⁵ For the discussion of “extensibility” of software, see Section 1.3.1 Chapter 1

If the first step is about acquiring quasi-*physical* discreteness for the closed source software, the second step can be seen as an attempt to define the *legal* discreteness through using proprietary software licensing schemes. Because the first step itself cannot prevent the hidden object code from being reverse engineered back into a source code version by its users, programmers thus design an extra layer of protection by using proprietary software licences. These licences are usually drafted in a way that a broad, and often overbroad, range of activities by users—including reverse engineering, copying, modifying, redistributing—are strictly prohibited.¹⁶ In short, the purpose of these proprietary licences is to make the software as tightly discrete a product as possible in legal terms, though some of the prohibitions may risk upsetting the balance preset by copyright law.¹⁷ Furthermore, software as discrete product is often released via a legal vehicle of standard-form contracts on a take-it-or-leave-it basis. In this way, the traditional manifestation of *consent* through bargained-for exchanges is minimised to a level where the non-negotiable licensing terms start to resemble those unmodifiable physical features imbedded in the software products. Radin incisively observes that these standard form licences undermine traditional consent as the centrepiece of contracts. She points that the “contract-as-consent” model is being replaced with the “contract-as-product” model, where licensing terms are an inseparable part of the discrete product.

In this [contract-as-product] model, the terms are part of the product, not a conceptually separate bargain; physical product plus terms are a package deal. The fact that a chip inside an electronics item will wear out after a year is no less and no more a feature of the item and its quality than the fact the terms that come with the item specify that all disputes must be resolved in California under California law. In this model, unseen contract terms are no more and no less significant than unseen internal design features; and it is not remarkable

¹⁶ Mark Lemley, “Terms of Use” (2006) 91 *Minnesota Law Review* 459

¹⁷ For example, UK copyright law does allow reverse engineering or “decompilation” for the purpose of achieving interoperability by lawful users without copyright holders’ permission. Any licensing term that attempt to contract out this permitted act would be illegal. See Section 50B, CDPA 1988; See also Section 3.3.3, Chapter 3 of this dissertation for more detail.

that there is no choice other than the take-it-or-leave-it choice not to buy the package.¹⁸ (added emphasis)

In the US, the contract-as-product model becomes the dominant approach to software licensing after the US Seventh Circuit's case *ProCD v. Zeidenberg*.¹⁹ It is explicitly endorsed by Easterbrook who argues that “[c]ontractual terms are product attributes—no different functionally from the quality of a car's tires, a TV's capacitors, or a software package's features [...]”.²⁰ Contrary to this view, I will later show, in Section 4.4, that licences that facilitates FOSS collaboration need to keep a critical distance from the “contract-as-product” model, but a vision of “contract as relation” is more appropriate to account for the real lived cooperative experience among FOSS programmers.

Presentation and Proprietary Software Licensing

Closely related to proprietary software licences' attempt to discretise software development process into separate non-extensible end-products is what Macneil calls the “presentation” of total obligation into these licensing documents. “Presentation”, in short, is a technique used by classical contract law to bring the future into the present.²¹ It is “a way of looking at things in which a person perceives the effect of the future on the present”.²² Classical contracts use this technique to reduce the uncertainty of contractual exchanges that may last for a period of time into the future.

Recall that ideally discrete transactions are assumed to be transient and short-lived as if they almost have no duration. However, this view of zero-duration transactions does not always tally well with reality, where most contractual exchanges do not consummate at one single moment, but last into the future. In order to cope with this problem, classical contract law has to employ the technique of presentation by compressing the future relation into a single point as if it had no duration at all. Macneil describes how presentation takes place:

¹⁸ Margaret Jane Radin, “Humans, Computers, and Binding Commitment” (1999) 75 *Indiana Law Journal* 1125 at 1126

¹⁹ See Michael J. Madison, “Legal-ware: Contract and Copyright in the Digital Age” (1998) 67 (3) *Fordham Law Review* 1025

²⁰ Frank Easterbrook, “Contract and Copyright” (2005) 42 (4) *Houston Law Review* 953 at 968

²¹ Macneil, *NSC*, p.60

²² Macneil, *supra* note 7 at 863

[Presentiation] is a recognition that the course of the future is so unalterably bound by present conditions that the future has been brought effectively into the present so that it may be dealt with just as if it were in fact the present. Thus, the presentiation of transaction involves restricting its expected future effects to those defined in the present, *i.e.*, at the inception of the transaction.²³

From proprietary software developers' point of view, the very indefinitely long-term collaborative relations under the hacker customs are indeed too open-ended to be predictable. Proprietary software developers have to use classical contract law's technique to presentiate total obligation into a licensing document. In this light, users' activities that may prolong the lifespan of transactions (e.g. through reverse engineering, user customisation, error corrections, redistribution etc.) are all deemed to be undesirable and they should be minimised under licensing terms in order to reduce future uncertainties.

4.2.2 Relational Approach: Projecting Exchange into the Future

The radically decentralised FOSS production and its licensing schemes defy classical contract law in two senses: they are neither discrete, nor can they be presentiated. First, FOSS is designed to be extensible and customisable, and it invites users to become co-developers to modify and improve the software wherever they see appropriate. Stallman observes that FOSS development is like “an evolutionary process, where a person would take an existing program and rewrite parts of it for one new feature, and then another person would rewrite parts to add another feature”.²⁴ From proprietary software developers' viewpoint, the indefinitely long “evolutionary process” is unwieldy and unmanageable, because it threatens the transactional discreteness that is more conducive to commercialisation of the software products. In contrast, FOSS developers see the long “evolutionary process” exactly as a strength that should be celebrated. Unlike short-lived discrete transactions, the non-discrete collaborative relations make FOSS projects capable of growing and perfecting for a considerable period of time. FOSS licences here play an

²³ *ibid.*

²⁴ Stallman, “Why Software Should Be Free” at <<http://www.gnu.org/philosophy/shouldbefree.html>>

important role in facilitating the collaborative efforts among programmers, though they are not collaborative relation itself.

Secondly, total presentation is very unlikely to take place in a radically decentralised environment where FOSS is produced. This is because the collaborative relations in FOSS projects are rather open-ended and improvisatory and it is impossible to presentiate future creative efforts onto one present paper or electronic document that is intended to binding. To some extent, a FOSS project can be likened to a marriage or a family business where “the participants never intend or expect to see the whole future of the relation as presentiated at any single time, but view the relation as an ongoing integration of behavior which will grow and vary with events in a largely unforeseeable future.”²⁵ Although there is no presentation of total obligations for FOSS contributors, this does not mean that there is no planning whatsoever at all within FOSS projects. Instead, a modicum of preliminary planning by a few lead programmers is always necessary to make sure that all contributions can later be safely and effectively pieced together into one coherent artefact under a modular architecture.²⁶ In other words, these projects do involve some level of planning, which prevents them from freewheeling into complete anarchy. However, this planning in FOSS projects is incremental and tentative, and it is not anywhere close to presentation.²⁷ It is up to a small group of lead developers to find out the right balance where partially presentiated obligations do not hurt the flexible and serendipitous nature of FOSS projects.²⁸

Relational Exchange and FOSS Collaboration

Classical contract law is centred around the problem of enforcing promises in discrete transactions. It asks whether a promise or a set of promises made by a party

²⁵ Macneil, “Restatement (Second) of Contracts and Presentiation”, (1974) 60 (4) *Virginia Law Review* 589 at 595

²⁶ See Section 1.3.1, Chapter 1

²⁷ On top of the preliminary design for FOSS projects, participants often involve in what Macneil calls “post-commencement planning”, where “planning continues after formation of the relation and after entry of a new person into the relation.” Macneil, *NSC*, p.27

²⁸ Kelty argues that coordination in FOSS project means privileging “adaptability” over “planning”: “This involves more than simply allowing any kind of modification; the structure of Free Software coordination actually gives precedence to a generalized openness to change, rather than to the following of shared plans, goals, or ideals dictated or controlled by a hierarchy of individuals.” Kelty, *Two Bits*, p.211

should be enforced or not in a discrete and presentiated exchange.²⁹ In contrast, relational contracts are anchored in the more flexible and less presentiated exchanges and they are of great importance to sustain long-term FOSS collaboration.

Macneil defines his conception of “contract” as “no more and no less than the relations among parties to the process of projecting exchange into the future”,³⁰ or more succinctly, “the projection of exchange into the future”³¹. This definition reveals a crucial distinction between the relational approach and the classical approach in terms of their respective attitudes towards the element of “futurity” in contract: A classical contract sees future as the source of uncertainty and unpredictability, which must be tamed by the classicist technique of total presentiation at the time when the contract is made. In other words, the total legal obligation of the classic contract is set at the beginning of the exchange when an offer meets its acceptance. In contrast, a relational contract is more flexible and adaptable to the future. Far from abhorring the element of futurity in exchange, relationalists regard future as a source of serendipity that could be celebrated. To put the contrast fully in another way: a classical contract presentiates the future into the present, while a relational contract projects the current exchange into the future. This contrast is also readily applicable to the difference between proprietary software developers and FOSS programmers. The former equate future with uncertainty that must be minimised at all cost, while the latter celebrate the serendipitous element of future that may unfold gradually in a radically decentralised creative environment.

Promissory and Nonpromissory Projectors

For Macneil, a relational contract is more than merely a matter of enforcing promises. It covers a broader scope than an explicitly bargained-for promissory exchange under a classical bilateral executory contract.³² A valid relational contract can have both

²⁹ For example, this promise-centred view of contract can be found in the US Restatement (Second) of Contract, which explicitly defines “contract” as “a promise or set of promises for the breach of which the law gives remedy, or the performance of which the law in some way recognizes as a duty.” American Law Institute, Section 1, Restatement (Second) of the Law of Contracts

³⁰ Macneil, *NSC*, p.4

³¹ Macneil, “The Many Futures of Contracts”, (1973-74) 47 *South California Law Review* 692 at 712-3

³² Whitford points out that Macneil’s works on RCT are more than a theory of contract, but they are very close to a general theory of social order. “The reader should be aware that Macneil himself

promissory and non-promissory aspects, both of which generate expectations for parties to project exchange into the future. To begin with, a “promise” is a most straightforward projector that parties use to make explicit verbalised agreement with others. Macneil defines “promise” as the “[p]resent communication of a commitment to engage in a reciprocal measured exchange” and it is an “extraordinarily powerful mechanism for projecting exchange into the future”³³. However, promissory projectors do not form the whole picture of contractual exchanges. From a relationalist perspective, there are also nonpromissory projectors that play an equally important role in shaping parties’ expectations.³⁴ For example, the previous course of repeated dealings unsupported by verbalised agreements can be important nonpromissory projectors from a relational perspective.³⁵ Macneil argues that nonpromissory projectors are important in shaping relational exchanges in both primitive and modern societies:

Nonpromissory exchange-projectors [...] come in a great many forms. In all societies, custom, status, habit, and other internalizations project exchange into the future. In some primitive societies these may be the primary projectors, with promise relating to exchange playing only a very minor role, if that. Moreover we err if we fail to recognize that such nonpromissory mechanisms continue to

conceives of his work as much broader than anything most other contract scholars recognized as contract law. His relational contract theory encompasses all exchange, and because Macneil sees exchange occurring almost everywhere, his theory becomes in effect a general theory of the social order.” William C. Whitford, “Ian Macneil’s Contribution to Contracts Scholarship”, (1985) *Wisconsin Law Review* 545 (hereafter “Macneil’s Contribution”)

³³ Macneil, *NSC*, 7

³⁴ In the academic literature, these nonpromissory projects are also known as the “implicit dimension” of contractual exchanges. For more detail about classical contract law’s limited efforts to accommodate the implicit dimension, see David Campbell and Hugh Collins, “Discovering the Implicit Dimensions of Contracts”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. By David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003)

³⁵ The English case *Baird v. Marks & Spencer* highlights the divide between the classical and relational approaches to the previous course of dealings as a nonpromissory projector. In this case, the claimant Baird had been a garment supplier to the defendant Marks & Spencer for the past thirty years without a written agreement. The latter suddenly stopped placing order from the former, which believed that they deserved reasonable notice before the proper termination of the relation. The UK Court of Appeal, from a classical perspective, disregarded the previous relation and refused to imply a contract between the two parties. *Baird Textile Holding Ltd. v. Marks & Spencer plc*. [2001] EWCA Civ 274.

In contrast, Mulcahy and Andrews, employing a relational analysis, finds that the non-verbalised agreement manifested in long-term relation should be enforceable. Linda Mulcahy and Cathy Andrews, “Baird Textile Holdings v Marks & Spencer Plc” in *Feminist Judgements—From Theory to Practice* (Oxford and Portland, Oregon: Hart, 2010)

play vital parts in the most modern and developed of societies. Even kinship, a form of status which plays major roles in so many societies, is by no means absent as an exchange-projector in [modern society], although it may now be overshadowed by class or other structures with partially related roles.³⁶

In the history of FOSS collaboration, the early hacker custom has long functioned as a nonpromissory projector influencing computer hackers' "relational" exchanges since the 1950s and 1960s. This hacker custom, which was only retrospectively documented by Steven Levy in 1984, was initially only "an ethic seldom codified, but embodied instead in the behavior of hackers themselves".³⁷ The advent of FOSS licences like GNU GPL largely verbalise the some of the previously uncoded programmers' stewardship responsibilities into express promissory projectors. Note that express licensing terms in FOSS licences only codify a minimum set of legal responsibility necessary for the preservation of software commons, but it by no means spells out every single detail in FOSS collaboration.³⁸ In short, non-verbalised commitments play an important role in the day-to-day operation of FOSS collaboration and their importance cannot be eclipsed by the verbalised licensing terms.

RCT's Two Implications for FOSS Licensing

Given the subject of this dissertation, it is impossible to give a comprehensive survey of Macneil's RCT and its great influence on contemporary contracts scholarship.³⁹ I narrow my research down to two implications that are most crucial to a relational analysis of FOSS licensing. These two implications are based on a general survey of Macneil's theoretical contributions to contracts scholarship by Whitford, who concludes that Macneil has two important "messages" to lawyers. The first message is relatively well received and the second is still largely underappreciated by lawyers.

³⁶ Macneil, *NSC*, p.7

³⁷ Levy, *Hackers*, p.7

³⁸ In fact, it is impossible for general-purpose FOSS licences like the GPL to go beyond specifying the minimum responsibilities. For example, important issues concerning when, what, by whom and how those many contributions are made often remain unspecified in these FOSS licences.

³⁹ For a recent research on Macneil's great contribution to contract scholarship, see Cathy Joanne Andrews, *Bridging the Divide—An Exploration of Ian Macneil's Relational Contract Theory and Its Significance for Contract Scholarship and the Lived World of Commercial Contract*, PhD Thesis, (London: Birkbeck College, University of London, 2010)

Macneil's first message is that "there is no single moment at which the parties confirm a meeting of the minds respecting the important terms of the contract."⁴⁰ This is clearly an attack on classical contract law's insistence that there should be a single grand moment where parties' minds meet and all their obligations would be fully presentiated from that moment on. In a more realistic fashion, RCT suggests that parties' consent to an agreement may well be reached incrementally through a period of time. Whitford finds that this insight has been "generally accepted": in this sense, RCT has superseded classical contract law and becomes the "now mainstream contract theory."⁴¹ In the US context, the recent judicial development in the more specific area of information product licensing has largely proved Whitford's observation correct. The line of jurisprudence spearheaded by the 1996 landmark *ProCD* ruling again serves as an important example for the purpose of this chapter. In *ProCD*, Easterbrook challenged the classicist model by suggesting that a licensing contract was not formed at the single point when the software was purchased. Instead a user's consent to the licence can be constructed in a *period* of time between the point of purchase and the actual use of the licensed product. In this period, the user arguably had ample opportunity to read and digest the content of the licence and there was no excuse for him to say no consent was formed. Barnett, in approving this logic behind *ProCD*, comments that "[t]here is no reason in principle why contracts cannot be formed in stages, provided the circumstances or prior practice makes this clear or adequate notice is provided. This insight is neither revolutionary nor reactionary."⁴² From a pure classicist point of view, *ProCD* and its progeny represent a "neoclassical" turn⁴³ that started to erode the traditional mechanism of consenting where minds only meet at single one point. This move is sometimes lamented as the

⁴⁰ The single moment of the meeting of minds is exactly what is required by the classicists' technique of presentation: "Before this grand meeting of minds, there was no contractual liability. And after this point, all important decisions—particularly the determination of the terms governing the relationship and the measurement of expectation damages—could be reached only by referring to that all encompassing agreement." Whitford, "Macneil's Contribution" supra note 32, at 546

⁴¹ *ibid.*, at 548

⁴² Randy E. Barnett, "Consenting to Form Contracts", (2002) 71 *Fordham Law Review* 627 at 644; Macaulay looks at the same issue from a slightly different angle. He points out that Barnett's proviso (i.e. "provided the circumstances or prior practice makes this clear or adequate notice is provided") in the above quoted sentence matters should be given more weight in reality: "The essential part of Barnett's observation is in his proviso. Those who slip terms into the fine print almost never make anything clear or give notice adequate enough to serve the legitimating idea of actual or manifest choice." Stewart Macaulay, "Freedom from Contract: Solutions in Search of a Problem?" (2004) *Wisconsin Law Review* 777, FN 94 at 804 (hereafter "Freedom from Contract")

⁴³ See *infra* Section 4.4.1

cause leading to “waning of consent”⁴⁴ or even “death of assent”⁴⁵ in the neoclassical standard-form licensing jurisprudence, which is much less strict about the affirmative manifestation of licensees’ explicit assents. Although the neoclassical law seems to take on board Macneil’s first message, I call for a more careful and nuanced reading of RCT, which will reveal that “assent” or “consent” is not irreversibly “dead”, but they still play “a vital triggering mechanism” of relational exchanges⁴⁶ and it is important to understand “consent” in relational terms. Just as Gudel observes there has never been a real “decline of assent,” but there is only “a decline of assent *discretely* understood” in contract jurisprudence.⁴⁷

Macneil’s second message is a cautionary one: when contracts scholarship moves away from the consent-driven classical model, it is in danger of recalibrating contract law as a neoclassical apparatus solely for the purpose of promoting parties’ “desire to maximize wealth”.⁴⁸ This is not what a true relational approach should embrace, but in fact “parties in relational contracts frequently temper wealth maximization goals with other objectives.”⁴⁹ In fact, Macneil himself makes it clear that people joining in relational exchange are by no means *solely* motivated by maximising their individual utility, but they are also driven by their desire for enhancing social solidarity.⁵⁰

Going back to the discussion of standard-form software licensing, the neoclassical *ProCD* ruling again is exactly an exemplary occasion, against which Macneil’s second message warns. Sidelining the role of classicist “consent”, Judge Easterbrook reoriented the ground for enforcing standard-form licences towards the need of

⁴⁴ Margaret Jane Radin, “Boilerplate Today, The Rise of Modularity and the Waning of Consent”, (2006)104 *Michigan Law Review* 1223

⁴⁵ Mark Lemley, “Terms of Use” (2006) 91 *Minnesota Law Review* 459 at 464

⁴⁶ Macneil, *NSC*, p.50; I will also show later that, for those FOSS products targeted at non-sophisticated end users (rather than professional co-developers), a higher requirement of affirmative manifestations of users’ assents (e.g. through the clickwrap technology) is not completely unnecessary.

⁴⁷ Paul J. Gudel, “Relational Contract Theory and the Concept of Exchange”, (1998) 46 *Buffalo Law Review* 763 at 773

⁴⁸ Whitford , “Macneil’s Contribution” *supra* note 32 at 549; For more detail about the this wealth maximisation view of relational contract, see for example, Charles Goetz and Robert E. Scott, “Principles of Relational Contracts” (1981) 67 (6) *Virginia Law Review* 1089

⁴⁹ Whitford, *supra* note 32 at 550

⁵⁰ Ian R. Macneil, “Exchange Revisited: Individual Utility and Social Solidarity”, (1986) 96 (3) *Ethics* 567; Robert W. Gordon, “Macaulay, Macneil, and the Discovery of Solidarity and Power in Contract Law” (1985) *Wisconsin Law Review* 565

reduction of transaction costs.⁵¹ It is necessary to enforce the license in this case, because it would “make information more readily available, by reducing the price ProCD charges to consumer buyers.”⁵² The kind of consent obtained through traditional ways of bargaining would only make transactions too costly to benefit information product suppliers and consumers economically.⁵³ It is important to know that an economic justification of standard forms like this was not first invented by Easterbrook, nor is it unique to mass software licensing, but it is preceded by many non-software cases long time ago before the *ProCD*. Whitford nicely summarises this general shift from obtaining classicist consent to guaranteeing wealth-maximisation as the basis of legitimating Standard-Form Contracts [SFK]:

It is now generally recognized that true consent to all aspects of the SFK is usually lacking. From a wealth maximisation perspective, this is as it should be. Individual negotiation of every contractual detail would take too much time. Because of the absence of true consent, a majority of commentators no longer regard agreement to a SFK as sufficient to validate its content. Rather, judicial and legislative oversight of some terms is deemed both appropriate and desirable. In suggesting ways to exercise that oversight, however, commentators very often look just to wealth maximization values. The question they frame is what terms the parties would have agreed to if they had negotiated the contract, were well informed, and were concerned solely with wealth maximization.⁵⁴ (internal citations omitted)

With this background in mind, I argue that any serious attempt to justify FOSS licensing must cautiously distance itself from the wealth-maximisation oriented jurisprudence developed in *ProCD*. Instead, it needs to take on board Macneil’s second message that relational contract promotes a multiplicity of values but the

⁵¹ According to Macneil, neo-classical contract law is built on the foundation of microeconomics. The Coasean transaction cost analysis may subtly vary this neoclassical model, though the presupposition that human beings are utility-maximisers does not change. See Macneil, I.R., “Economic Analysis of Contractual Relations: Its Shortfalls and the Need for a Rich ‘Classificatory Apparatus’ ”, (1981) 75 *Northwestern University Law Review* 1018

⁵² *ProCD v. Zeidenberg*, 86 F.3d 1447 at 1455

⁵³ In Easterbrook’s own words, this would “drive prices through the ceiling or return transaction to the horse-and-buggy age.” 86 F.3d 1447 at 1452

⁵⁴ Whitford, *supra* note 32 at 553,

wealth-maximisation objective is only one of them.⁵⁵ The reduction of RCT to the neoclassical model would only risk impoverishing the Macneilian relational contract. A radically decentralised FOSS project attracts a large number of volunteer contributors rightly because utility maximisation is *not* the only predominant motivational force, but programmers are motivated by a variety of reasons. A 2005 empirical survey shows that the top three motivations behind volunteer FOSS contribution are intrinsic intellectual enjoyment of coding, the prospect of improving programming skills and the belief that FOSS is a worthy cause for its own sake. None of them is directly about the increase of contributors' material utility or wealth.⁵⁶ It can be argued that relational contracts in FOSS projects are intended to create a kind "relational wealth"⁵⁷, which splices together a variety of motivational values, which can be either utilitarian or non-utilitarian. It is a kind of non-monetary wealth that is absent in *ProCD*, where the plaintiff and the defendant are in antagonistic competition and have no intention of collaborating to build a common project. Macneil's contract norm of "preservation of the relation" therefore is of critical importance in maintaining the relational wealth in FOSS projects and it could be a new basis for evaluating the legal strength of FOSS licensing.⁵⁸ With Macneil's two messages in mind, I will move on to explore how FOSS licences are understood by existing doctrinal rules as a contrast to the relational perspective in the following section.

4.3 Three Doctrinal Routes to Enforcing a FOSS Licence

Are FOSS licences contracts? If so, are they enforceable contracts? If not, according to what other possible legal doctrines might they be enforced? Should the application of a chosen doctrine take into account of the relational aspect of FOSS licensing

⁵⁵ Whitford believes that "Macneil's many descriptions of relational contracting illustrate that parties to such contracts commonly pursue a number of objectives, only one of which is wealth maximization." Whitford, *supra* note 32 at 560

⁵⁶ I will discuss the three motivations in more detail in Chapter 5 on FOSS authorship. See Karim R. Lakhani and Robert G. Wolf, "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects", in *Perspective on Free and Open Source Software*, eds. by Feller, Fitzgerald, Hissam & Lakhani (Cambridge, Mass.: MIT Press, 2005)

⁵⁷ For a general account of relational wealth, see Romesh Diwan, "Relational Wealth and the Quality of Life" (2000) 29 *Journal of Socio-Economics* 305; in the specific context of this chapter, I wish to show that FOSS projects' "relational wealth" is the kind of peer-produced "wealth" as indicated in Benkler's book title *The "Wealth" of the Network*. Although "relational wealth" may indirectly bring material wealth for FOSS programmers, the former cannot be entirely reduced to the latter.

⁵⁸ For the norm of "preservation of the relation" in contractual exchanges, see Macneil, *NSC*, p.66

practice? As FOSS licences are designed to facilitate decentralised collaboration, there is no doubt that it is important to find answers to these questions. However, given the complex nature of the issue, there is little consensus on the enforceability of these licences among scholars.⁵⁹ There are at least three different theories, or three doctrinal routes, under which that a FOSS licence may be enforced. The first theory suggests that a FOSS licence can be enforced if it is a valid contract. The second theory argues that a FOSS licence may well lack a contractual status due to problems such as the lack of bargained-for exchanges. Instead, it should be enforced as a bare licence (or pure property licence) regardless of the licence being contractual or not. The third theory finds that the second theory has a weakness: a bare licence is revocable by the licensor. So in order to prevent a FOSS licensor from going back on his promise, the equitable doctrine of estoppel might be evoked when necessary.

4.3.1 First Route: Contractual Licence

The first route argues that a FOSS licence such as GNU GPL can be enforced as a contractual licence. One of most enthusiastic champions of this argument is Gomulkiewicz, who believes that the GPL fulfils all the requirements under a US model code known as the Uniform Computer Information Transaction Act (UCITA) to be a contract.⁶⁰ The UCITA explicitly recognises a “license” as having a contractual status because here “licence” is defined as “a contract that authorizes access to, or use, distribution, performance, modification, or reproduction of, informational rights, but expressly limits the access or uses authorized or expressly grants fewer than all rights in the information, whether or not the transferee has title to a licensed copy.”⁶¹ However, this resort to the UCITA is problematic because the

⁵⁹ Nimmer even suggests that to ask whether a FOSS a licence is a contract or not is a wrongly framed question in the first place. It simplifies the issue too much. The right way of approaching the issue is to look into how these FOSS licences are actually used or intended to be used. Raymond Nimmer, “Is the GPL license a Contract? The Wrong Question” 5 September 2005 at <<http://www.ipinfoblog.com/archives/licensing-law-issues-is-the-gpl-license-a-contract-the-wrong-question.html>>

⁶⁰ Robert W. Gomulkiewicz, “How Copyleft Uses License Rights to Succeed in the Open source Software Revolution and the Implications for the Implications for Article 2B” (1999) 36 *Houston Law Review* 179

⁶¹ Section 102 (a) (41), UCITA

spirit of this Act has its provenance in the hugely controversial *ProCD* ruling⁶². In the FOSS community, Richard Stallman is openly against the UCITA, which is believed to suit proprietary software developers' interests but not FOSS programmers.⁶³ Largely for this reason, the Free Software Foundation has refused to categorise the GPL as a UCTIA-type contract.⁶⁴ However, in order to do some justice to FSF's argument later, I first need to give the licence-as-contract argument some benefit of the doubt at the moment by examining how a "contract" may be formed in a "licence".

Formation of Contract

Anglo-American contract law insists on three main components being present to form a contract: offer, acceptance and consideration. If a FOSS licence has to be recognised as having a contractual status, then it must have all these three components.

(A) "Offer" in FOSS Licensing

It is a relatively straightforward issue to find an "offer"⁶⁵ in a FOSS licence. An offer is a licensor's manifested willingness to give users permissions to access, use, modify or redistribute a piece of FOSS and these permissions are usually accompanied by some restrictions pursuant to Free Software Definition and Open Source Definition. Rosen points out that the willingness to offer can be manifested by posting the software to a publicly accessible FOSS repository website on the

⁶² UCITA is said to be the "bad fruit" from the ProCD jurisprudence. Roger C. Bern, " 'Terms Later' Contracting: Bad Economics, Bad Morals, and a Bad Idea for a Uniform Law, Judge Easterbrook Notwithstanding", (2003-2004) 12 *Journal of Law and Policy* 641 at 772

⁶³ Stallman points out that the UCITA is a product of proprietary software developers' lobbying efforts and it run against the spirit of free software movement. See Stallman, "Why We Must Fight UCITA", 31 January 2000 at <http://w2.eff.org/IP/UCITA_UCC2B/20000131_fight_ucita_stallman_paper.html>

⁶⁴ See infra subsection 4.3.2

⁶⁵ In an US context, an offer is defined as the "manifestation of willingness to enter into a bargain so made as to justify another person in understanding that his assent to that bargain is invited and will conclude it." American Law Institute, S.24. Restatement (Second) of Contracts

internet (e.g. SourceForge) so that “all prospective licensees will be able to retrieve the software under the terms of the license”.⁶⁶

(B) “Acceptance” in FOSS Licensing

According to the classical contract model, an acceptance should be the “mirror image” of the offer, that is, it must be “absolute” and must “correspond with the terms of the offer”.⁶⁷ The offeree needs to unequivocally convey his intention to accept “without leaving room for doubt as to the fact of acceptance, or as to the coincidence of terms of the acceptance with those of the offer”.⁶⁸ An offeree may accept an offer through verbalised agreements, but he may also manifest his acceptance through non-verbal forms of conduct, which is not unusual in the mass-market off-the-shelf software world.⁶⁹ There are three main ways that software licensing terms may be offered to potential licensees for acceptance—shrinkwrap, clickwrap and browsewrap—each of which will be discussed in turn.

Shrinkwrap and the ProCD case Early mass-market software products are often sold in boxes wrapped with shrinkable clear plastic under the so-called shrinkwrap or box-top licences. When the purchasers pierce the plastic open, it is normally assumed that they assent to the licensing terms attached to the software within the boxes. Before 1996, shrinkwrap licences were routinely found to be unenforceable at the federal appellate level across the US.⁷⁰ However, in 1996, the situation was changed by the US Seventh Circuit’s landmark decision in *ProCD v Zeidenberg*, where Judge Easterbrook ruled that a shrinkwrap license was enforceable as a contract.⁷¹ In this case, the plaintiff *ProCD* sells a database product called SelectPhone(TM), comprising a national telephone directory and a software program for searching the

⁶⁶ Lawrence Rosen, *Open Source Licensing—Software Freedom and Intellectual Property Law*, (Upper Saddle River, NJ: Prentice Hall PTR, 2005) p.60

⁶⁷ J. Beatson, *Anson’s Law of Contract* (Oxford: OUP, 2002, 28th Edition) p. 37

⁶⁸ *ibid.*

⁶⁹ Of course, the offeree may also choose not accept the offer. For instance, in a Scottish case, a customer refused to accept the offer by returning the software to the seller without opening the shrink-wrapped box. See *Beta Computers (Europe) v Adobe Systems (Europe)* (1996) SLT 604

⁷⁰ See *Vault Corp. v Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988); *Step-Saver Data Sys. Inc. v Wyse Tech*, 939 F.2d 91 (3rd Cir. 1991); *Arizona retail v. Software Link*, 831 F. Supp. 759 (D. Ariz. 1993)

⁷¹ Scholars normally divide the ProCD ruling into two portions: one on contract formation and the other on copyright. In this subsection I only focus on the former but I will go back to the latter in Section 4.4 of this chapter.

data. The product is delivered on CD-ROM disks shrinkwrapped by a notice: “Both the software and the data listings are subject to the terms and conditions of the enclosed license agreement which is part of this product and printed in full on the enclosed envelope. Please read fully the license agreement.”⁷² Note that this notice itself is not the actual licence containing the terms and conditions, but the former is merely an alert to the latter. It is impossible for the defendant-purchaser Matthew Zeidenberg to know, let alone consent to, the content of the licence when he purchased the boxed ProCD product.

Since Zeidenberg did not make any explicitly verbalised assent to the licence that he was only able to view later after the purchase, the court faced a difficult question as to whether the licence was actually “accepted” by, and then binding on, the purchaser.⁷³ Judge Easterbrook ruled that Zeidenberg did “accept” the licence and he was thus bound by it. The reason is that although Zeidenberg did not make a verbal assent to the licence in the classical sense, his “acceptance” of the licence can be *inferred* from his failure to return the ProCD product to the vendor after reading the licence. Easterbrook bases this judgment primarily on Section 2-204(1) of the Uniform Commercial Code (UCC) that relaxes the classicist offer-acceptance doctrine by allowing assents to be manifested “in any manner sufficient to show agreement”.⁷⁴ In this sense, a buyer’s acceptance does not have to be obtained after a conventional bargaining process, but it can be indicated by any conduct of “performing the acts the vendor proposes to treat as acceptance.”⁷⁵ Because the ProCD vendor “proposed a contract that a buyer would accept by *using* the software after having an opportunity to read the license at leisure” (original emphasis), Zeidenberg’s actual use of the software should be construed as his assent to the

⁷² According to Macaulay, this notice “was printed in six-point type in the middle of a long paragraph on the bottom flap of the software box.” Stewart Macaulay, “Freedom from Contract”, supra note 42 at 805

⁷³ This problem is not a peculiar only to this case, but it is generic to all shrinkwrap licences: “Unlike a typical unilateral contract, in which one party accepts an offer by engaging in conduct that unmistakably indicates assent—say, painting my house—the conduct used as evident of a shrinkwrap contract is hardly unambiguous evidence of assent.” Lemley, “Terms of Use”, supra note 16, at 468

⁷⁴ The whole clause of S2-204 reads: “A contract for sale of goods may be made in any manner sufficient to show agreement, including conduct by both parties which recognizes the existence of such a contract.” However, Macaulay finds that Easterbrook’s use of S2-204 does not really “follow the definitional cross reference to Section 1-201 (b)(3), which says that an ‘agreement’ is ‘the *bargain* of the parties in fact [...]’ ” (added emphasis) Macaulay, “Freedom from Contract”, supra note 42, FN 103 at 806

⁷⁵ 86 F.3d 1447 at 1452

licence and he had “no choice, because the software splashed the licence on the screen and would not let him proceed without indicating acceptance.”⁷⁶

The ProCD ruling has provoked tremendous controversy in the academic world. Eric Posner observes that the decision has “precipitated a typhoon of academic hostility” and remains “probably the most criticized case in modern history of American contract law”.⁷⁷ What is most disturbing to contract scholarship is that ProCD overtly challenges the consent-driven classical contract model, which relies on bargained-for exchanges to generate full and unambiguous meeting of the minds. Post finds it is hugely problematic for Easterbrook to use Section 2-204 of the UCC to construct Zeidenberg’s inactivity as the “tacit assent” to the ProCD licence:

Even if we were to use the ‘gestalt’ approach to contract formation in S. 2-204 [of UCC] which would look at all the communications between the parties without an attempt to isolate a particular document or communication that was the offer or the acceptance, there is still the problem of finding assent in the passivity of the buyers unless we are willing to assume ‘tacit assent’ from their silence or inaction. That too flies in the face of traditional contract doctrine.⁷⁸

Concurring with Post’s critique, Macaulay also finds Easterbrook’s interpretation has stretched a little too much the concept of “assent” (as “in any manner sufficient to show agreement” under S.2-204) to possibly cover unbargained or under-bargained non-consensual relations. This would allow manufacturers of consumer goods such as the ProCD supplier to gain unilaterally a wide range of “freedom *from* contract” by “packing inside the box contract clauses that attempt to repeal various laws that business dislike.”⁷⁹ From the viewpoint of a classical “contract purist”, it is just “very

⁷⁶ Note also that Zeidenberg was given the opportunity to return the product if he found the licensing terms were unsatisfactory to him. *ibid.*, 1452

⁷⁷ Eric Posner, “*ProCD v Zeidenberg* and Cognitive Overload in Contractual Bargaining” (2010) 77 *The University of Chicago Law Review* 1181 at 1193

⁷⁸ Deborah W. Post, “Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook”, (2000) 16 *Touro Law Review* 1205, FN64 at 1226

⁷⁹ Steward Macaulay further comments that to assume consumers’ assents to licensing terms from their failure not return the products is only “a bad joke”. See Macaulay, “Freedom from Contract”, *supra* note 42 at 805

difficult to offer a convincing argument that these hidden clauses work to create a contract with the desired effect”.⁸⁰

The *ProCD* decision is important in the sense that it forcefully kick-starts a neoclassical reengineering of the classical contract law in software licensing jurisprudence. By sidelining the strictly bargain-oriented classical doctrine, Easterbrook shifts to an economical justification of standard-form licensing couched in Coasean language of transaction cost.⁸¹ From a neoclassical point of view, the traditional fully-dickered bargaining is not a viable option in the mass-market software world because it would only slow down the transactional speed and blur the discreteness of transactions.⁸² In the *ProCD* case, without enforcing the attached standard-form licence, it would otherwise “drive prices through the ceiling or return transaction to the horse-and-buggy age” even the terms could only be viewed after the purchase.⁸³ In summary, the great significance of *ProCD* lies in the fact that Easterbrook articulates a dominant neo-classical rationale for software licensing jurisprudence despite the incessant academic resistance to it. The rationale mandates that routine enforcements of standard-form information product licences is a predominant economic necessity for increasing market efficiency and decreasing transaction cost in the mass-market software world, while customers’ manifestation of assents to these licences is demoted to be a secondary issue, which should be as flexible as possible. This rationale by Easterbrook has twofold consequences. On the one hand, it clears the classicist hurdle to enforcing standard-form licences under contract law. On the other hand, the dominance of the *ProCD* rationale also impoverishes the software licensing jurisprudence where the possibility of non-

⁸⁰ *ibid.*, at 805-6

⁸¹ For a detailed Coasean explanation of the main rationale behind *ProCD* ruling, see Frank Easterbrook, “Contract and Copyright” (2005) 42 (4) *Houston Law Review* 953

⁸² For example, Raymond Nimmer is an important champion of this view. He argues that fully dickered agreement is a romantic view about contract formation and it does not apply to mass-market software licensing. “Under contract law, formation of a contract and definition of its terms do not require sophisticated or equally leveraged parties, nor parties with incentive to devote time, effort, and cost to negotiate. Standard terms, leverage, and adherence to pre-set terms characterize all commerce. Most importantly, it has never been considered to be the role of contract law to generally reshape the balance created by market conditions. [...] The ability of parties to standardize and control the terms under which a product or information is marketed gives an important element of efficiency in transactional environment.” See Raymond T. Nimmer, “Breaking Barrier: The Relationship between Contract and Intellectual Property Law” (1998) 13 *Berkeley Technology Law Journal* 827 at 847

⁸³ 86 F.3d 1447 at 1452

neoclassical alternatives risks being ignored and underdeveloped. To fill this gap, I will analyse a *relational* rationale for FOSS licensing in Section 4.4.

Clickwrap and Browsewrap Given the high penetration of fast broadband Internet connection among the population worldwide, FOSS nowadays is more likely to be downloaded directly from a repository website to the local computers rather than being delivered on the physical medium of CD-ROMs using a shrinkwrap.⁸⁴ In fact, clickwrap and browsewrap licences are more frequently used by FOSS and they raise slightly different concerns in terms of contract formation. Clickwrap licences require affirmative actions from licensees to manifest their acceptance: they are asked to press the button “Yes, I Agree” as a way of assenting to the licensing terms and conditions before they actual download or install the software. Kim has observed that click-wrap licences “do not raise the same contract formation concerns as shrinkwrap agreements because the user typically has notice of the terms and has an opportunity to read them *prior* to engaging in the contractual relationship.” (original emphasis)⁸⁵ The clickwrap technology employs a slightly more licensee-friendly measure by using an interactive interface, which usually is a pop-up dialogue box displaying the licensing terms. Software users are thus notified of the existence of the licence, though they may not actually read or understand everything in it. FOSS projects, especially those are keen to rapidly build a non-sophisticated end-user base, often employ clickwrap as its licensing interface.

Although clickwrap is a slightly more user-friendly technology than shrinkwrap, this does not mean the problem of assumed “assents” to licensing term is completely gone. In fact, FOSS software developers often make users to manifest “acceptance” in two ways: “acceptance” can be manifested *not only* through clicking the “Acceptance” button *but also* through the actual conduct of installing and using the software. For example, when Google released its FOSS browser “Chrome” in

⁸⁴ A FOSS project is often a work in progress and it may distribute many beta versions in a very short space of time before releasing a stable deliverable version. This process requires frequent bug-fixing and rewriting the code. So it is not always necessary to transfer the ever-changing code onto a CD, when it can be easily downloaded from a repository website such as Freshmeat.net and SourceForge.net.

⁸⁵ Nancy S. Kim, “Clicking and Cringing”, (2007) 86 *Oregon Law Review* 797 at 842-843

September 2008⁸⁶, Google Chrome Terms of Service assumed users to accept the licence through either of the two ways:

2.2 You can accept the Terms by:

(A) clicking to accept or agree to the Terms, where this option is made available to you by Google in the user interface for any Service; or

(B) by actually using the Services. In this case, you understand and agree that Google will treat your use of the Services as acceptance of the Terms from that point onwards.⁸⁷ (see *Figure 4.1 Screenshot of Google Chrome Terms of Service*)

Note that the clickwrap here is lumped together with actual use of software as assumed manifestation of assents⁸⁸. In this light, a user may give his assumed assent to the licence even though he fails to click the “Accept” button, which means the clickwrap mechanism may be bypassed completely.⁸⁹

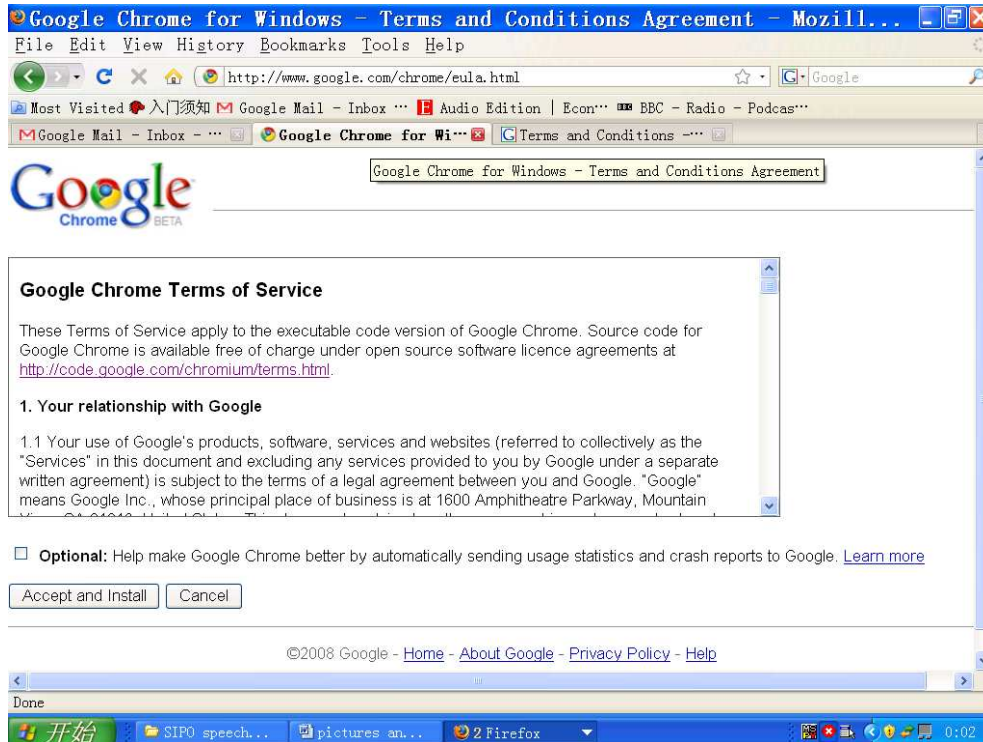
⁸⁶ Chrome is released under two twined licences: one is the BSD licence for the source code release and the other is the Terms of Service for consumer executive code release. The latter uses the clickwrap technology and makes reference to the former.

⁸⁷ Section 2, Google Chrome Terms of Service

⁸⁸ *c.f.* The ProCD vendor also assumes actual use of the product as assent to the licensing terms.

⁸⁹ For a hypothetical example, a university student is using a Chrome browser to surf internet in a computer lab. Because the software has already been installed by the university IT staff, this student does not have to manifest his assent by clicking any button and he thus does not know the licensing terms. However, he would still be assumed to assent the Chrome licence because of his conduct of using the software. In this sense, it is possible for the clickwrap mechanism to be completely bypassed for this reason.

Figure 4.1 Click-wrap: Screenshot of Google Chrome Terms of Service



Note: This is a licence is for distributing the executable code version of Chrome and it should be used together with BSD License for distributing the source code version of the software.

Compared with shrinkwrap and clickwrap licences, browswrap licences are the most problematic of the three types. It assumes that by using or installing the software, “licensees” automatically “agree” to the terms and conditions that can be viewed somewhere as a webpage or merely an electric notice. Not all browswrap licences had been enforced by courts after *ProCD*⁹⁰. As a rule of thumb, in order to decide whether a browswrap licence is valid or not, it is important to know if the browsrapped software carries *prominent notices* for a user to be aware of the licensing terms. In other words, “a user is not bound by a [browswrap] contract of which he is not made aware.”⁹¹ This means that the licensing terms should be reasonably easy and straightforward to be located and read by users. For example, in the case *Ticketmaster Corp. v. Tickets.com, Inc.* the court ruled that the “terms of use” on the plaintiff’s webpage was enforceable, because there was evidence

⁹⁰ Lemley observes that courts tend to enforce browswrap against businesses but not against individuals. See Lemley, “Terms of Use” (2006) 91 *Minnesota Law Review* 459 at 476

⁹¹ Christian H Nadan, “Open Source Licensing: Virus or Virtue?” [2002] *Texas Intellectual Property Law Journal* 349 at 364

showing that the defendant had actual knowledge of it.⁹² In contrast, in *Specht v. Netscape Communications Corp.*, a disputed arbitration clause in a browserwrap licence was ruled to be unenforceable, because users was not given a prominent notice about the existence of the licensing terms on the defendant’s webpage.⁹³

FOSS developers must put a lot of emphasis on notifying their licensees about the licensing terms when the browserwrap is used. For instance, Section 5 of the GPL v.3 makes it clear that all downstream distributors of modified source versions have the responsibility of giving “prominent notices” about the licensing status of the code involved:

- [...]
- b) The work must carry *prominent notices* stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to ‘keep intact all notices’.
- [...]
- d) If the work has interactive user interfaces, *each must display Appropriate Legal Notices*; however, if the Program has interactive interfaces that do not display Appropriate Legal notices, your work need not make them do so.⁹⁴ (added emphasis)

Note that Section 5 (d) contains a proviso that GPL programmers are not required to use “interactive interfaces” to display appropriate legal notices. The consequence is that GPLed works are allowed to be conveyed without using the clickwrap technology. This is in line with the “technology-neutral requirement” stipulated in the Open Source Definition (OSD). In 1998, Open Source Initiative amended the OSD by adding a tenth criterion making sure that non-clickwrap technologies (including those with no graphic interface at all) should not be discriminated against in distributing FOSS. The official Rationale attached to Criterion 10 “License Must Be Technology-Neutral” in the OSD explains:

Rationale: This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. *Provisions mandating so-called “click-wrap” may conflict with*

⁹² 2003 U.S. Dist. LEXIS 6483 (C.D. Cal. 2003)

⁹³ 306 F.3d 17 (2d Cir. 2002)

⁹⁴ Section 5. “Conveying Modified Source Versions”, GPL v.3; similarly, Section 4 of the GPL requires to carry an “appropriate copyright notice” conveying a verbatim copy of the covered work.

important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that **(a)** redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that **(b)** the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.⁹⁵

(C) Consideration

The last leg of contract formation—consideration—seems to be an even more unsettled issue in FOSS licensing. Under the doctrine of consideration, common law courts do not generally enforce a simple donative promise,⁹⁶ but only enforce one party’s promise that is reciprocated with another party’s promise or performance. In *Currie v. Misa*, Lush J. points out that a “valuable consideration, in the sense of the law, may consist in some right, interest, profit, or benefit accruing to the one party, or some forbearance, detriment, loss, or responsibility given, suffered, or undertaken by the other.”⁹⁷ In short, a consideration must confer some benefits and detriments to the promisee and promisor.⁹⁸

A consideration must have some value, though common law cares very little about how valuable it needs to be. Treitel points out that “an act, forbearance or promise will amount to consideration only if the law recognises that it has some *economic values*” and it “may have such value even though the value cannot be precisely quantified.”⁹⁹ So when it comes to “non-monetary performance of doubtful economic value to the promisor”, it becomes a difficult issue to decide whether it can be

⁹⁵ OSI, “The Open Source Definition (Annotated)”, Version 1.9 <<http://www.opensource.org/docs/definition.php>>

⁹⁶ Melvin Eisenberg, “Donative Promises” (1979) 47 (1) *The University of Chicago Law Review* 1

⁹⁷ *Currie v. Misa* (1975) L.R. 10 Ex. 153 at 162

⁹⁸ Atiyah finds that the doctrine of consideration contains “two legs” 1) “the idea that a promise is legally binding if it is given in return for some *benefit* which is rendered, or to be rendered, to the promisor”; and 2) “a promise becomes binding if the promisee incurs a detriment by reliance upon it, that is, if he changes his position in reliance on the promise in such a way that he would be worse off if the promise were broken than he would have been if the promise had never been made before.” He also points out that the second leg as detrimental reliance has “close connections with other branches of the law, such as the law of tort, and also various equitable doctrines, as well as the doctrine of ‘estoppel’ in its various forms”. Atiyah, *An Introduction to the Law of Contract*, pp,118-119

⁹⁹ Guenter Treitel, *The Law of Contract*, (London: Sweet and Maxwell, 2003, 11th Edition) p.83

qualified as the right kind of consideration recognised by law¹⁰⁰. In most FOSS projects, volunteer licensees' contributions are mostly non-monetary performances (e.g. reporting bugs or testing submitted patches etc.) and it is not always clear whether these performances can have the right "economic values" to qualify as consideration as defined by Treitel above.

In this scenario, now I try to explore a question that is often asked: does the GNU GPL involve a valid consideration with the right economic value to form a binding contract? The scholarly community again fails to reach a consensus and is divided into two camps on the issue. The first camp believes there is consideration in the GPL. This view is championed by Wacha who believes that there are reciprocal "mutual promises" between licensors and licensees. The licensors offer the software under certain conditions, while the licensees, "as consideration, agree[] to keep all copyright notices intact, to insert certain required notices, and to redistribute code only under certain conditions."¹⁰¹ The second camp believes that the GPL fails to be a contract for a lack of consideration. This view is held by Kumar, who looks at the same set of restrictions that was examined by Wacha in the GPL, but she reaches a diametrically opposite conclusion. According to Kumar, consideration is exactly GPL's "Achilles heel". The licensors' offer of software is a kind of conditional donative promise and the licensees' adhering to the attached conditions is not a consideration to this offer: "The GPL places a number of restrictions on the user of GPL-licensed software [...] However, adhering to restrictions on the use of a licensor's copyrighted software is not consideration because the restrictions do not directly benefit the licensor."¹⁰² In other words, there is no reciprocal exchange between the licensors and licensees, because the former do not get any a clear benefit in return. Kumar then argues that the GPL is "based on real property licenses", which concurs with the FSF's official explanation of the GPL as a bare licence. "Suppose that a landowner grants a revocable license to the public to cross through a strip of the landowner's property to access a public beach. The landowner does not explicitly receive anything in return from the public. Though the landowner may limit the

¹⁰⁰ Mindy Chen-Wishart, *Contract Law* (Oxford: OUP, 2008, 2nd Edition) p.134

¹⁰¹ Jason B. Wacha, "Taking the Case: Is the GPL Enforceable" (2005) 21 *Santa Clara Computer and High Technology Law Journal* 451 at 474 (internal citations omitted)

¹⁰² See Sapna Kumar, "Enforcing the GNU GPL" 2006 *University of Illinois Journal of Law, Technology and Policy* 1 at 19-21

public's access to certain times of day, these 'burdens' on the public do not serve as consideration for using the landowner's property. They are merely limitations on the access that the public is receiving."¹⁰³

Following Kumar's argument, the debate between the two camps can be also framed as the one about whether licensees' obeying the conditional restrictions under the copyleft is a benefit to the licensors or not. Or to put it another way, is the relation between the licensors and licensees under the copyleft a reciprocal one? It could be argued that Kumar slightly over-simplifies the issues. In fact there are two groups of GPL licensees. The first group simply uses the software and makes no publicly released contributions back to the community. These licensees do not benefit the licensors.¹⁰⁴ In contrast, the second group do not only modify software and but they also choose to share the source code of these modifications with the community. This use of the GPLed software is different from the situation of "real property licences" to give public access as discussed by Kumar above. The second group of licensees is not merely passively allowed public access to the GPLed code, but they also proactively make *improvements* to the code. The improved code will thus bring some benefit to the whole community including the original licensors, though this benefit may not be the type that can be immediately converted into monetary wealth. Of course, these two groups of licensees may not remain mutually exclusive and things can change. Some licensees in the first group may choose to join the second group and become proactive contributors over the time and vice versa.

4.3.2 Second Route: Bare Licence

As has been discussed above, the attempt to use the first route to enforce FOSS licences as contract are likely to encounter two uncertainties: 1) lack of explicitly verbalised assents from users and 2) lack of consideration understood by classical contract law. The second route attempts to bypass these two difficult issues by treating a FOSS licence as a bare licence, which is a unilateral permission given by the property owner to enable the licensees to use the work in a way which would

¹⁰³ Kumar, *ibid.*, at 20-21

¹⁰⁴ GPL allows private modifications that are not conveyed to the public. See Section 3.5.1 Chapter 3

otherwise be infringing.¹⁰⁵ The idea of “bare licence” is a relatively unfamiliar concept to software licensing jurisprudence, because it is usually discussed in the land or real property context.¹⁰⁶ It is interesting to see that the old doctrine of “bare licence” initially used in land law is now being revived in the FOSS world. This attempt is championed by Free Software Foundation (FSF), which interprets that the GPL is a bare licence but not a contract.¹⁰⁷ According to them, the permission under the GPL is unilaterally granted to licensees, which seems to be a one-way operation. Unlike a classical contract where an offeree needs to unequivocally “accept” an “offer”, licensees of the GPL as a bare licence are not required to verbally “accept” the licence. Because all GPLed software is copyrighted in the first place, one would have infringed the copyright without the permission from its owner. In other words, to obey the terms of the GPL is the condition of using the GPL covered work. Section (9) of the GPL v3.0 makes it clear that the “acceptance” in a classical contract is not required in the GPL as a bare licence:

You are not required to accept this License in order to receive or run a copy of the Program. [...] However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.¹⁰⁸

Eben Moglen, in the first International Conference on the GPL v3.0 (intending to clarifying FSF’s jurisprudence behind this new version of the GPL to the community), reiterates two points supporting FSF’s official position that the GPL is not a contract. First, bargained-for exchanges do not exist in the GPL. Second, the

¹⁰⁵ According to Rosen, if there is a failure of offer, acceptance or consideration to form a contract, a FOSS licence may fall back on copyright for being a bare property licence. See Rosen, *Open Source Licensing*, pp. 65-66

¹⁰⁶ The “bare licence” in land law is an old institution, which dates back to a seventeenth-century English case, where Vaughan C.J. defined a property licence simply as a permission that “only makes an action lawful, without which it had been unlawful.” *Thomas v. Sorrell*, (1673) Vaugh 330 at 351; see also I.J. Dawson and Robert A. Pearce, *Licences Relating to the Occupation or Use of Land* (London: Butterworths, 1979) p.1

¹⁰⁷ Moglen and Stallman, “Transcript of Opening session of first international GPLv3 conference”, transcribed by Ciaran O’Riordan, 16 January 2006 at <<http://www.ifso.ie/documents/gplv3-launch-2006-01-16.html>>

¹⁰⁸ Section 9, GPL 3.0

GPL is a copyright licence, without which one's use of the GPLed software would lead to copyright infringement.

We [on behalf of the FSF] have not argued now, nor will we, nor can anyone argue, who reads the text of the language, that the receipt of the code is some quid-pro-quo for the acceptance of some terms. [...] arguments based on the contractual exchange of the code for promises of compliance have nothing to do with us. We give permissions here and the enforcement weight of our license lies in the fact that you have no permission to propagate, that is, you have no permission to do what copyright law requires permission to do, but through this license. That's our legal theory and we are sticking to it.¹⁰⁹

Furthermore, there are also two policy reasons why the FSF insists that the GPL is not a contract but a bare licence. First, FSF's position has to do with its attempt to avoid the unpopular model contract code UCITA, which derives its jurisprudence from the controversial *ProCD* ruling.¹¹⁰ The UCITA that treats software licences as contracts, according to Stallman, is essentially a product of proprietary software lobbying efforts.¹¹¹ By arguing the GPL is a bare licence, the free software movement keeps a critical distance from the legal theory behind the UCITA-type contract law. Second, Moglen argues that contract laws in different countries around the world are by no means uniform, and it would be difficult for the new globally applicable GPL v3.0 to handle the diversity within the world contract regimes.¹¹² When the GPL is a bare licence, it will base its validity solely on software copyright. Because most countries' copyright laws are modelled upon the same set of international agreements such as the Berne Convention, it is more conceivable to reconcile approaches than when dealing with world contract laws. Even with a single country like the US where different states have their own contract laws (while the

¹⁰⁹ Moglen and Stallman, "Transcript of Opening Session of First International GPLv3 Conference", supra note 107

¹¹⁰ Note Easterbrook treats "licences" are "ordinary contracts accompanying the sale of products" but he deliberately leave the question "why licences are contracts" unanswered: "[w]hether there are legal difference between 'contracts' and 'licenses' [...] is a subject for another day." *ProCD v. Zeidenberg*, 86 F.3d 1447 (7th Cir.1996) at 1450

¹¹¹ Stallman, "Why We Must Fight UCITA", supra note 63

¹¹² GPL 2.0, when it was first written, was only intended to be used within the US. The popularity of this licence around the world makes the FSF decide that the new GPL 3.0 should be globally applicable.

Copyright Act is federal), the many state contract regimes can be unwieldy for individual licensors and licensees alike in the US to handle.¹¹³ For this reason, the enforcement of the GPL would be better off when treated as a non-contractual bare licence:

The reason that's our legal theory and [we] are sticking to it remains the one we gave before. There are [too many] contract law schemes in the world and the more you depend upon them, the more variability you will have. Berne [Convention] is good, the harmonisation of copyright is good, for us. Our rules will use a toolset that is as close to global standard as we can get.¹¹⁴

Moglen's concern is understandable that a big diversity of contract regimes around the world would Babelise the jurisprudence behind the GPL and reduce the legal certainty for enforcing the licence in a global context. However, this point is not universally accepted. The fragmentary contract regimes may have been exaggerated. Rosen points out that today's globalised economy has required a high level of "consistency of commercial transactions" and "contracts are interpreted in much the same way around the world."¹¹⁵ At the same time one should not underestimate inconsistency of the copyright regimes around the world. Again, Rosen argues, for example, that there is no agreed definition of "derivative work" in global copyright law, and its meaning vary from country to country. So it would be better to have the licence drafters to clarify its meaning through the vehicle of contract rather than merely relying on copyright.¹¹⁶

It is also important to note that FSF's reliance on Berne-type copyright law is mainly a pragmatic choice and it is just for the convenience of licence enforcement. The hacker custom is still against any private property regime including copyright. (Recall that Moglen's call for "[a]bolition of all forms of private property in ideas" in the dotCommunist Manifesto.¹¹⁷) Stallman followed up Moglen's above quoted

¹¹³ Although there is a uniform model contract law concerning computer information transactions, i.e. UCITA, its impact is very limited. Only two states—Maryland and Virginia—have ratified UCITA so far.

¹¹⁴ See "Transcript of Opening Session of First International GPLv3 Conference", supra note 107

¹¹⁵ Rosen, *Open Source Licensing*, p.58

¹¹⁶ *ibid*, p.58

¹¹⁷ Moglen, The dotCommunist Manifesto, January 2003, <<http://emoglen.law.columbia.edu/publications/dcm.html>>

speech by making their hackers' ideological leaning clear that they are not uncritically endorsing the global copyright regime:

That [‘Bern is good, the harmonisation of copyright is good, for us’] doesn’t mean that we are in favour of copyright law as a general matter. [...] We're not totally against copyright law, in a simple or blanket sense either, but we're not defending the global copyright system that has mostly been imposed on the world merely because we use it because it's there. [...] We are not endorsing the Berne plus WTO system of copyright law as it stands as a good thing, but it exists and whatever harm it may do in other areas, we're trying to do some good with it when we can.¹¹⁸

Revocability of Bare Licences

One of the most obvious weaknesses of a bare licence is that it is only binding on the licensee but not on the licensor. A bare licence can be unilaterally terminated or revoked at the pleasure of the licensor. In other words, a FOSS licence as a bare licence is not mutually binding. This problem might lead to unfairness when the licensee has contributed modified source code back to the project or merely formed reliance by using the licensed software. The US case *Microsystems Software, Inc. v. Scandinavia Online* has exposed this problem.¹¹⁹ In this case, two computer hackers, Eddy Jansson and Matthew Skala, developed a program called CPHack which was released under the GNU GPL. CPHack was designed to disable Microsystems’s censorware known as “Cyber Patrol 4”. The two hackers were then sued by Microsystems for their anti-censorware. In order to settle the dispute, the two hackers agreed to revoke the GPL and assign the copyright of CPHack to Microsystems. So the copylefted software thus became proprietary.¹²⁰ The GPL, when it is a bare licence, cannot prevent the licensors from going back on their promised permissions

¹¹⁸ Supra note 107. Stallman’s explanation also echoes the main argument of 2 Chapter that free software programmers endorse “IP” law only to the extent it can be leveraged to facilitate FOSS collaboration.

¹¹⁹ *Microsystems Software, Inc. v. Scandinavia Online* AB, 98 F. Supp. 2d 74 (D. Mass., 2000), aff’d, 226F. 3d 35(1st Cir., 2000)

¹²⁰ “The Story of CPHack”, at <<http://cphack.robinlionheart.com/#slapp>>(Last updated: 21 June 2002)

made in the licence. The unilateral withdrawal of the GPLed code understandably would pose another layer of uncertainty to any FOSS project.¹²¹

4.3.3 Third Route: Promissory Estoppel

The second route does not make obligations in a FOSS licence mutually binding. In other words, a bare licence can be enforced only against the licensee but not the licensor. In order to compensate this weakness, a third route via an equitable doctrine of “estoppel” is suggested to prevent FOSS licensors from revoking or terminating the licence at will when the licensee has clearly developed a detrimental reliance upon it.¹²² Unlike the doctrine of consideration in contract law that enforces bargained-for exchanges, estoppel can be used to enforce reliance-based liability arising from unreciprocated promises.¹²³ Some scholars have already proposed this third route to enforce FOSS licences (including the GPL) in the US context,¹²⁴ where the doctrine of “promissory estoppel” is codified in Section 90, Restatement (Second) of Contracts:

A promise which the promisor should reasonably expect to induce action or forbearance on the part of the promisee or a third person and which does induce such action or forbearance is binding if injustice can be avoided only by

¹²¹ McGowan observes that the revocability of the GPL makes FOSS programmers inclined to make compromise with proprietary software companies in a dispute like this one. See David McGowan, “Legal Implications of Open-Source Software”, (2001) *University of Illinois Law Review* 241, FN 283 at 302

¹²² In general, Denning finds that the equitable doctrine of estoppel “is a principle of justice and of equity” and it deals with a situation like this: “when a man, by his words or conduct, has led another to believe in a particular state of affairs, he will not be allowed to go back on it when it would be unjust or inequitable for him to do so.” Denning MR in *Moorgate Mercantile Co. Ltd. v. Twitchings* [1976] 1 QB 225, CA, at 241

¹²³ It is important to know that estoppel is often said to be “suspensive” rather than “extinctive”. This means that the promisor can be allowed to go back on their promise if the promisee has not yet developed reliance. See Beatson, *supra* note 67, p.117

¹²⁴ For example, Madison argues that “[i]f the author attempted to take the improved version of the code private, equitable theories such as estoppel might provide a useful backstop in cases where the facts could not support a formal contract theory.” Michael Madison, “Legal Implications of Open-Source Software” (2001) *University of Illinois Law Review* 241 at 297; Other scholars champion this approach includes Kumar, “Enforcing the GNU GPL”, *supra* note 102; Rosen, *Open Source Licensing*, pp.64-65; Chip Patterson, “Copyright Misuse and Modified Copyleft: New Solutions to the Challenges of Internet Standardization”, (2000) 98 *Michigan law Review* 1351, at 1379;

enforcement of the promises. The remedy granted for breach may be limited as justice requires.¹²⁵

As promissory estoppel protects software users' detrimental "reliance", so it is necessary to ask what constitutes "reliance" in the FOSS context.¹²⁶ As a rule of thumb, users' conduct of modifying the original FOSS would suffice to constitute reliance. Kumar argues that the mere use of the software will be difficult to prove reliance. However, if users make derivative works based on the FOSS licensed software, it is a strong sign that reliance has been established and his "reliance" should be protected against original software developers' attempt to terminate the GPL.¹²⁷

Although the promissory estoppel in the FOSS context has never been actually tested in court, the danger of software developers' termination of a FOSS licence is not entirely hypothetical. There is good strategic reason for a commercial company to make their software released under a FOSS licence in the beginning and later terminate it to back-claim royalties when they see appropriate. To spell it out, a company may strategically adopt a FOSS licence and then let its user-base grow because of the generous grants in this licence. When this FOSS licensed software becomes so popular that it turns out to be the *de facto* standard, it will be too costly for its users to switch to other software. At this point, the company may threaten to terminate the FOSS licence and start to collect royalties from users. If users refuse to pay back royalties, they can then choose to terminate the licence and then sue for copyright infringement.¹²⁸ In this scenario, promissory estoppel can be a useful doctrinal tool to be evoked to prevent users against this kind of strategic use of FOSS licences.¹²⁹

¹²⁵ ALI, Section 90, *Restatement (Second) of the Law of Contracts*

¹²⁶ Cooke observes that "reliance" is at the heart of estoppel, but it is also fundamentally a matter of "causation", which is very difficult to be proved. So there "is a generous helping of common sense" involved to decide what is reliance and causation. Elizabeth Cooke, *The Modern Law of Estoppel* (Oxford: OUP, 2000) p.105

¹²⁷ Kumar, *supra* note 102 at 25

¹²⁸ This worry has been raised by Chip Patterson, "Copyright Misuse and Modified Copyleft: New Solutions to the Challenges of Internet Standardization", (2000) 98 *Michigan Law Review* 1351

¹²⁹ However, it is worth noting that promissory estoppel cannot be used as a cause of action but it can only be used as a defense as if it a "shield" under English law. see *Central London Property Trust Ltd. v High Trees House Ltd.*, [1947] KB 130

4.4 Conceptualising the GPL as a Relational Contract

Having canvassed the difficulties of applying existing doctrinal rules to FOSS licensing above, I will now try to re-examine the GPL from a relational perspective.¹³⁰ The re-examination is divided into two subparts. Firstly, in Section 4.4.1, I identify two obstacles to re-conceptualise GPL as a true Macneilian relational contract. One can straightforwardly be derived from the consent-driven classical contract law, whilst the other comes from the more insidious neoclassical *ProCD*-type law. Secondly, in Section 4.4.2, I propose that GPL is better understood as a relational “umbrella agreement”, which is designed to harness the serendipitous nature of the peer production of FOSS and at the same time stabilise the long-term collaborative relation.

Before I start, a caveat is worth making at the start. A paper document on which the GPL is written cannot be a relational contract. The GPL may become relational only when it is adopted and used by certain a FOSS project (e.g. the Emacs project or the Linux project) where there is an ongoing collaborative relation. To make it clear, my focus of this section is not about the GPL being merely as a textual document as such, but it concerns how the GPL is relationally understood in the context of the real lived collaborative experience as in peer-produced FOSS projects.

4.4.1 Two Obstacles: Classical and Neoclassical Laws

There are two conceptual obstacles to developing a new line of enquiry about the GPL in relational terms. A relationally understood GPL needs to overcome each of them before it can potently account for the collaborative relations in a decentralised and coordinated FOSS project. The first obstacle lies in the inconvenient fact that the GPL is a standard-form licence, which often does not require unequivocally verbalised manifestation of consent from the licensees. This is at odds with classical contract law that is anchored in explicitly bargained-for exchanges. Under the classical model (also known as the contract-as-consent model), a thoroughly dickered bargain gives rise to a meeting of the minds at a single moment when the offeree

¹³⁰ This enquiry continues and deepens the discussion of “GPL and FOSS Collaboration” as in Section 3.5 of Chapter 3.

accepts the offer. Only at that single point the total legal obligation is presentiated and becomes fully binding on the parties.

In order to negotiate this first obstacle posed by classical contract-as-consent law, RCT provides a more realistic and sophisticated understanding of the role of explicit “consent” in contract formation. From a relational perspective, consent does not simply take place at one stroke, and nor does it always have to be fully verbalised. Instead, consent often happens *incrementally* over a period of time, and it may well come out of a mix of verbalised and non-verbalised commitments. Macneil points out that the exercise of choice in a consensual relation is “an incremental process in which parties gather increasing information and gradually agree to more and more as they proceed.”¹³¹ The incremental process of consent can hardly come out of an isolated discrete transaction but it has to be through a series of repeated dealings between parties in a timeline. As the gradual formation of consent through repeated dealings clearly does not fit into the discrete transactional model, a more nuanced understanding of the “bargaining” mechanism is warranted. Lon Fuller famously observes that either full bargain or zero-bargain is very rare in real contractual exchanges. Between these two extremes, people often reach agreement through “half-bargaining”: “where men cannot bargain with words, they can often half-bargain with deeds; tacit understandings arising out of reciprocally oriented actions will take the place of verbalized commitments.”¹³² In a long-term FOSS project, there are plenty of chances to “half-bargain with deeds” in a rich collaborative relation, which can often alleviate the heavy burden on the one-shot explicit consent as the sole device to effectuate the obligations in a licence like the GPL. In contrast, a consumer of proprietary software is much less likely to develop an incremental consent, because a proprietary software developer normally has no intention to enter into a collaborative relation with his consumer and he would thus tries to make a software transaction as discrete as possible. As there is no future relation beyond this particular discrete transaction, it is impossible for the user of the proprietary product to incrementally assent to the licensing terms over a period of time.

¹³¹ Macneil, “Economic Analysis of Contractual Relations: Its Shortfalls and the Need for a Rich ‘Classificatory Apparatus’ ”, (1981) 75 *Northwestern University Law Review* 1018 at 1041

¹³² Lon Fuller, “The Role of Contract in the Ordering Processes of Society Generally” (originally written for the third edition of *Basic Contract Law*) in *The Principles of Social Order*, edited by Kenneth I. Winston (Durham, N.C.: Duke University Press, 1981) p.185

The second obstacle to conceptualising the GPL as a relational contract does not come directly from the classical model, but from the more elusive neoclassical contract law, which assumes the supremacy of market efficiency in maximising individuals' material utility. It is not uncommon to confuse the neoclassical model with Macneil's relational contract model, though the two models may bear certain surface resemblance in terms of their tinkering with the classic model.¹³³ However, the neoclassical model does not substantially deviate from the classical law's ideology which assumes the supremacy of the goal of maximising individual utility. In fact, neoclassicism is even more decisive and ruthless in pursuing and executing the discreteness of transactions than classical contract law. Macneil observes that the extreme pursuit of transactional discreteness makes the neoclassicism unsuited to deal with the relational aspect of contracting:

Neoclassical contract law is founded in theory and organization on the discrete transaction, but with many a relational concession. It can often deal adequately with the more discrete issues in contractual relations. But when discrete and relational principles conflict, neoclassical law lacks any overriding relational foundation, and thus lacks a resource often needed in relational law.¹³⁴

In the software licensing context, the neoclassical approach is probably good enough to supply an economic justification of proprietary software licensing that promotes discrete transactions, but it is rather incompetent in explaining non-market-based relational exchange in FOSS collaboration. There is a conspicuous "relational" gap to be filled by a true Macneilian understanding of relational contract, where exchanges are not solely conducted under the goal of maximising individual utility.

As it is impossible to canvass the whole picture of neoclassical contract law in this chapter,¹³⁵ I only limit my discussion to the neoclassicist strategy to replace

¹³³ For example, both neoclassical law and RCT would relax the classicist offer-acceptance doctrine in contract formation.

¹³⁴ See Macneil, *NSC*, p.72

¹³⁵ For a detailed account of the subject, see Macneil, I.R., "Economic Analysis of Contractual Relations: Its Shortfalls and the Need for a Rich 'Classificatory Apparatus' ", (1981) 75 *Northwestern University Law Review* 1018. Also, Macneil indicates that the neoclassical law is based on neoclassical economics. Campbell observes: "Macneil somewhat confusingly uses the term 'neoclassical' to capture both the neo-classical economics that are the foundation of the classical law of contract and the neo-classical law that he believes is an improvement on that law (i.e. it is a distinct

“consent” with “market efficiency” as the main rationale in legitimating standard form contracts. This strategy is achieved by two steps: first, it dismisses thoroughly negotiated “consent” as unrealistic and unnecessary. The second step then recalibrates the legitimation mechanism against a new benchmark of market efficiency, which becomes the most salient feature of neoclassical contract law. Raymond Nimmer succinctly records this two-step shift from classicism to neoclassicism as follows:

[...] the [classical contract’s] ideas of choice and agreement convey[] a romantic view of contracts, i.e., that the choices must follow actual negotiation between parties of equal bargaining power. Negotiation over terms seldom occurs in either a mass market or a commercial marketplace. Our economy, and the mass market in particular, is not, and never was, a bazaar economy characterized by recurrently dickered terms shaped to each transaction. Nor can it ever be so. Economics and efficiency concerns preclude it.¹³⁶

From a neoclassical point of view, it is clear that fully-dickered negotiations are unnecessarily costly, but they must give way to the goal of maximising parties’ individual utilities in the most calculably efficient fashion. Easterbrook’s *ProCD* ruling epitomises this neoclassical view where the enforcement of the standard-form licence in dispute is based on the ground of economic efficiency in promoting discrete transactions of the ProCD product. As has already been shown in above Section 4.3.1, Easterbrook disregards the classicist offer-acceptance doctrine but favours the neoclassical UCC’s approach¹³⁷ that allows consent to be assumed “in

form of traditional contract law from the classical law [...]. The classical law is, of course, typically expressed in formal doctrinal terms to which, as has been said, Macneil does not pay particular attention. However, he describes writing on contract by law and economics scholars as the elaboration of the economic foundation of the classical law. ” David Campbell, “Ian Macneil and the Relational Theory of Contract” in Ian Macneil, *The Relational Theory of Contract: Selected Works of Ian Macneil*, ed. by David Campbell, (London: Sweet & Maxwell, 2001) p.29

¹³⁶ Raymond T. Nimmer, “Breaking Barrier: The Relationship between Contract and Intellectual Property Law” (1998) 13 *Berkeley Technology Law Journal* 827 at 846

¹³⁷ It is important know that UCC is exactly a neoclassical legislation that deviates from classical law to reflect complex mass-market transactions in American society. Post comments: “The U.C.C. was drafted and adopted in the post World War II era as the United States became a ‘consumer economy characterized by mass consumption, modern consumption and mass culture.” Deborah W. Post, “Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook”, (2000) 16 *Touro Law Review* 1205 at 1214

any manner sufficient to show agreement”.¹³⁸ Precluding time-consuming negotiations that traditionally legitimates consensual relations, he reduces the practice of standard-form licensing to an efficient money-saving device that is believed to be economically beneficial to consumers.¹³⁹ Post finds Easterbrook’s neoclassic strategy to trade consumers’ consent for market efficiency is a “ruthless” one. Easterbrook assumes that “reduced cost is what all consumers want”, but fails to appreciate “the values and desires (not translatable into dollars and cents) that animate human beings”.¹⁴⁰ The *ProCD*-type law is a slippery slope that may eventually lead to an undemocratic assent-destroying result: “Constructive assent, manufactured through the manipulation of the rules of contract formation and the interpretation of silence as assent (because it is read by the light of judge’s belief that he knows what is best for the consumer or for the economy), is inappropriate and undemocratic.”¹⁴¹ Post’s view is endorsed by Macaulay¹⁴², who in his own examination of *ProCD* case, openly favours Macneil’s approach over Easterbrook’s: “I like Ian Macneil’s approach much better than Easterbrook’s. Macneil concedes that [standard form contracts] are not real contracts, but he would enforce many of them. He argues that we enter many relationships where we do not know all the terms—marriage, the military, and jobs at university law schools. Our relationship with our computer or software supplier is just one more.”¹⁴³

Here Macaulay means “Ian Macneil’s approach” by the one advocated in the essay “Bureaucracy and Contracts of Adhesion”, where Macneil proposes a *relationally* understood “consent” to standard forms.¹⁴⁴ Unlike Easterbrook who ostensibly bases his judgement on a constructive “consent” (assumed from a consumer’s inactivity, i.e., his failure to return the product before use), Macneil is candid that the classical contractual consent, which is “individual manifestation of a willingness to be bound in relatively specific ways”, cannot work well in legitimating standard from

¹³⁸ S2-204, UCC.

¹³⁹ Easterbrook, in concluding his ruling, argues that “[e]nforcement of the shrinkwrap license may even make information more readily available, by reducing the price ProCD charges to consumer buyers.” 86 F.3d 1447 at 1455

¹⁴⁰ Deborah W. Post, “Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook”, (2000) 16 *Touro Law Review* 1205 at 1230-1

¹⁴¹ *ibid.*, at 1238

¹⁴² For Macaulay’s endorsement of Post’s essay on Easterbrook’s jurisprudence as just quoted above, see Stewart Macaulay, “Freedom from Contract”, *supra* note 42 at FN 86 at 802

¹⁴³ *ibid.* at 807

¹⁴⁴ Macneil, “Bureaucracy and Contracts of Adhesion”(1984) 22 *Osgoode Hall Law Journal* 5

contracts.¹⁴⁵ Scholars have to face the fact that it is impossible for adhering parties to comprehend every single detail in a standard form contract at the outset of a contractual relation. For Macneil, a consent to a standard form has to be understood as *a consent to join an unfolding relation* that is projected into the future.

Liberal society has always recognized numerous legitimate *relations* into which entry is by consent, but the content of which is largely unknown at the time the consent was given. This is the idea of joining a relation. We can join a law firm or a university faculty or any other employment relation; we can join the army; we can join a corporation by buying its shares; we can join in holy matrimony. In each instance we can do so in spite of large-scale ignorance about the restraints we are accepting. In spite of our ignorance liberal society will bind us to those unknown restraints.¹⁴⁶

Macneil finds that the “consent to join an approved relation”, though ubiquitous in liberal society, lacks open recognition by liberalist thinking and is largely “hidden in the liberal intellectual closet”.¹⁴⁷ In fact, any FOSS licence like the GPL deliberately does *not* presentiate the total legal obligation concerning making contribution to the project. So the consent to the GPL is largely a Macneilian *consent to join a relation*, which is gradually unfolding through ongoing collaborative efforts. This FOSS relation is not dissimilar from a marriage relation, which also normally avoids presentation of the total obligation but is more based mutual trust to carry on the relationship among the couple. I will soon show that GPL is in fact a constitution-like umbrella agreement that only promulgates general rules for those who are willing to *join the collaborative relation*¹⁴⁸ and its legitimacy is supported by Benkler’s peer production model.¹⁴⁹

¹⁴⁵ *ibid.*, at 5

¹⁴⁶ *ibid.*, at 20-21

¹⁴⁷ Macneil observes: “mainstream liberal thinking avoids the existence of relations like the plague, because the concept of relation, particularly when it is given the common label of status, is anathema to the individualism upon which liberalism is based.[...] Legitimation by consent to relations approved by society is, whenever possible, hidden in the liberal intellectual closet.” *ibid.*, at 21

¹⁴⁸ This view is corroborated by Weber’s observation: “The core constitutional message of an open source license is fashioned as a statement to the developers. And the foremost statement is that they will be treated fairly if they *join the community*.” (added emphasis) Weber, *Success*, p. 179

¹⁴⁹ See for detail *infra* subsection 4.4.2

Although the neoclassical model (unsatisfactorily) removes the classicist hurdle in contract formation, its own rationale to promote market efficiency in furthering individual utility maximisation becomes a new hurdle to a relational understanding of FOSS licences. The economic rationale behind *ProCD* is probably good enough to justify a proprietary software licence that is used for a one-shot discrete transaction, but it is hugely inadequate in coping with the more complex relational exchange in FOSS collaboration, where contributors are not single-mindedly motivated by the sole desire to maximise their individual material wealth. Whitford worries that Macneil's message of respecting a multiplicity of values held by participants in a relational contract would be unfairly overridden by the neoclassicist's sole agenda for wealth maximisation. He urges that "the law, and the legal academics, should more fully recognize the place of other values, especially *participation*, where a [standard form contract] is used in a relational setting. While individual negotiation of each contract may be just too inefficient, there may be other ways to provide adhering parties a sense of participation in framing the contents of their agreements" (added emphasis).¹⁵⁰ Note that Whitford here highlights "participation" as one of the values that can not be managed well in discrete transactions, but I think that this is exactly the value that should be taken seriously in FOSS collaboration. Whitford summarises the difference between discrete transaction and relational contract in terms of "participation":

Participation is another value commonly reflected in the behavior of parties to relational contracts. Participation, as I use the term, means that the parties seek influence in formulating the substantive content of a transaction. [...] In discrete transactions, take-it-or-leave-it bargains seem quite satisfactory because the party not drafting the terms can exercise effective control over its own well-being and, indirectly, over the terms of the standard form contract simply by declining to enter the transaction or refusing to enter another one. As transactions become relational, however, withdrawal becomes a less viable

¹⁵⁰ See Whitford, "Macneil's Contribution", *supra* note 32 at 553-4

means of control, and the parties seek direct participation the formulation of the rules of the relationship.¹⁵¹

“Participation” is an important site where a relational FOSS licence is distinguished from discrete transactional proprietary software licence. In a discrete transactional proprietary licence, not all parties can equally “participate” to decide the substantive content of a contract. The parties with stronger bargaining power often proffer the total legal obligation for the weaker parties, who may only choose to take it or leave it. In contrast, a relational FOSS licence does not presentiate total obligation for the adhering parties, but it requires all parties to constantly and proactively “participate” to shape their own obligation in terms of making actual contribution to a project in an ongoing cooperative relation. I will use the GPL to substantiate this argument in more detail below.

4.4.2 GPL as an Umbrella Agreement: Balancing Flexibility with Certainty

Though the value of “participation” is an important one, it is not a straightforward task to apply it to GPL as a relational contract. The main difficulty lies in the fact that many standard-form contracts “involving consumers are used in transactions that are not highly relational.”¹⁵² So how does GPL as a standard form become a relational contract and how do users of GPLed software to “participate” to shape their own obligation in a collaborative relation? In order to answer this question, I propose to examine the issue at two levels. The first level is the GPL as an umbrella agreement, which specifies the participants’ *minimum* legal commitments to guarantee software freedom but leaves open the substantive contents of actual contributions. The second level contains a myriad of sub-agreements about

¹⁵¹ Whitford observes that although the value of participation does not exclude material wealth maximisation, it can be pursued for its own sake. “Though often participation serves the wealth maximization objectives of the parties seeking it, it can be and often is an objective independent of its wealth maximizing effects. People want some control over their own destiny, even if sheer obedience to the dictates of another would be more efficient.” Whitford, *ibid.*, at 552

¹⁵² Whitford observes that most standard form contracts are not relational, but there are exceptions: “An outstanding example of the use of a SFK in a relational setting is the real estate lease. The contract for a new car is also quite relational. The parties are committed to a long term relationship because of the extensive warranties in use today, particularly since it usually is not practical for the buyer to simply sell the car and buy another when confronted with a problem. In these contexts, I believe that concern for participation and other values different from wealth maximization should be reflected in the content of regulation.” Whitford, “Macneil’s Contribution”, *supra* note 32, FN30 at 553-4

programmers' actual volunteer contributions, which are impossible without these contributors' direct *participation*. These second-level obligations are not, and cannot be, fully presentiated in the umbrella GPL, whose main purpose is to make sure that all contributions can be later legally pieced together into one sustaining free software project. In short, the GPL as an umbrella at the first level is merely to facilitate the real lived collaborative experience happening at the second level. It makes sure all contributions are not merely isolated transactions, but they can be considered under the same relational umbrella. In contrast, a proprietary software licence like the *ProCD* user licence is more likely to be employed only for a few isolated transactions, and users are not expected to develop a long-term collaborative relation with the original product supplier to substantially improve the product. In other words, a transactional software licence is not underpinned by a long-term collaborative relation between the original software developers and its users, but it is only a device to regulate some one-shot transactions of software as discrete commodities.

In the business sector, the use of umbrella agreements is by no means rare, because they are very useful to stabilise lasting commercial relations under which a series of interrelated transactions may take place.¹⁵³ Mouzas and Furmston point out that umbrella agreements are generally “not concerned with immediate contractual decisions but rather they explicitly spell out the principles that guide future contractual decision.”¹⁵⁴ The main reason for employing umbrella contracts is largely due to the tremendous difficulty of presentiating the *total* obligation for parties at the outset of a long-term cooperative relation. As it is impossible iron out every single detail for an unfolding relation, an umbrella document is needed to specify some general principles that prevent that relation meandering into nowhere. In this sense,

¹⁵³ Mulcahy and Andrews summarise the useful role of the umbrella contract in long-term business cooperation: “[...] the umbrella contract performs an extremely useful function in long-term business relations. It provides a framework of clauses or ‘constituion’ which sets out the arrangements and norms which will govern the parties’ commercial relationship. It provides certainty regarding the conditions under which particular exchanges may take place and a platform for ongoing negotiation. The umbrella contract articulates what have been called a high order of shared conventions which comprise customary expected, legal and non-legal rules and principles.” Linda Mulcahy and Cathy Andrews, “Baird Textile Holdings v Marks & Spencer Plc” in *Feminist Judgements—From Theory to Practice* (Oxford and Portland, Oregon: Hart, 2010) p.200

¹⁵⁴ They define “umbrella contracts” as “private arrangements that provide a framework of clauses which regulate future contracts.” Stefanos Mouzas, and Michael Furmston, “From Contract to Umbrella Agreement” (2008) 67(1) *Cambridge Law Journal* 37 at 38

an umbrella contract is very close to a kind of “constitution”, which provides “certainty regarding the conditions under which exchanges may take place” and also “a platform for an on-going negotiation”:

In this way, umbrella agreements take the form of ‘constitutions’ of contracts. To view an umbrella agreement as a constitution may be relevant to problems of interpretation remembering Marshall C.J.’s famous injunction that ‘we must never forget that it is a constitution we are expounding.’ The validity and legitimacy of constitutions is based upon the evolution of consent among related actors over time. This consent articulates a high order of shared conventions which comprise customary, expected, legal, and often non-legal rules and principles.¹⁵⁵ (internal citation omitted)

In the academic literature, it is not uncommon that the GPL is also often seen as a community’s “constitution”, which echoes nicely the above view that an umbrella contract is a “constitution” for participating parties in a relation. In his study of open source software, Weber repeatedly mentions that an open source licence is not narrowly a legal document for a particular transaction, but it is “a de facto constitution” that determines the governance structure of a project. Weber argues:

Yet there is another way to see the license, as a de facto constitution. In the absence of hierarchical authority, the license becomes the core statement of the social structure that defines the community of open source developers who participate in a project. One way to manage complexity is to state explicitly (in a license or constitution) *the norms and standards of behavior that hold the community together*.¹⁵⁶ (added emphasis)

No secondary commentary can be more revealing than the text in the GPL itself, which indicates that the licence is a constitutional umbrella specifying “the norms and standards of behavior that hold the community together”. The preambular text of the GPL makes it clear that the licence is no more than codifying general rules for

¹⁵⁵ Mouzas and Furmston, *ibid.*, at 38-9

¹⁵⁶ Weber, *Success*, p.179; Weber reiterates the same idea later in the same chapter: “[...] the licence represents foundational beliefs about the constitutional principles of a community and evolving knowledge about how to make it work.” Weber, *Success*, p.185

guaranteeing software freedom in all publicly conveyed contributions in a collaborative relation. It reads:

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, *the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users*. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.¹⁵⁷ (added emphasis)

Not unlike all other umbrella agreements, the GPL is also designed to achieve a balance between two interrelated needs, which are 1) “the need to remain flexible” and 2) “the need for certainty and calculability” in an ongoing cooperative relation among parties.¹⁵⁸ Recall that in Chapter 1, I argue that “collaboration” in any successful FOSS project has two defining aspects: it is not only 1) radically decentralised but also 2) coordinated among a large number of contributors.¹⁵⁹ The first aspect concerning decentralisation corresponds to the need for flexibility, while the second aspect concerning coordination corresponds to the need of certainty and calculability. Now I will explain both aspects/needs in turn.

Firstly, GPL does not, and cannot, presentiate the total obligation that licensees should bear in a radically decentralised collaborative environment,¹⁶⁰ but the need for *flexibility* or *serendipity* precludes presentation. The substantive decisions to contribute (e.g. what, when and how to contribute) are completely left to individual volunteers themselves.¹⁶¹ In this light, the GPL as a relational umbrella contract

¹⁵⁷ Para. 2, Preamble, GPL v3.0; It is although worth noting that the FSF claims that the GPL is a constitution for the whole free software community: GPL “is the Constitution of the Free Software Movement” as “its goals are primarily social and political, not technical or economic.” See FSF, “GPL Version 3: Background to Adoption”, <<http://www.fsf.org/news/gpl3.html>>

¹⁵⁸ These two needs are identified by Stefanos Mouzas, and Michael Furnmston, “From Contract to Umbrella Agreement”, supra note 154 at 49

¹⁵⁹ Section 1.3.1, Chapter 1

¹⁶⁰ Note that the concept of relational “contract” is very different from UCITA’s definition of “contract” as “the total legal obligation resulting from the parties’ agreement as affected by this [Act] and other applicable law”. Section 102 (a) (17), UCITA

¹⁶¹ Kelly observes that project leaders like Linus Torvalds do not directly assign or solicit contributions. Programmers make contribution entirely of their own volition: “At no point were the

dovetails well with Benkler's peer production system, which refuses to specify the creative tasks for "peer" programmers. The peer production thus distinguishes itself from market and firms, both of which struggle to specify or presentiate as much as possible the objects to be produced via the price mechanism or managerial commands.¹⁶² It encourages individual programmers to pursue their own intellectual interests during the evolution of a project. This is based on the belief that human creativity is a highly individuated enterprise and only individuals themselves can best identify what they are most capable of doing. For this reason, Benkler argues that creative tasks cannot be fully specified or presentiated by classical contract:

[...] human intellectual effort is *highly variable and individuated*. People have different innate capabilities, personal, social, and educational histories, emotional frameworks, and ongoing lived experiences. These characteristics make for immensely diverse associations with, idiosyncratic insights into, and divergent utilization of, existing information and cultural inputs at different times and in different contexts. *Human creativity is therefore very difficult to standardize and specify in the contracts necessary for either market-cleared or hierarchically organized production.*¹⁶³ (added emphasis)

Secondly, those "highly variable and individuated" contributions by peers also pose some *uncertainties* when they are needed to be pieced together into a coherent free software project. Without an explicitly agreed commitment, it is not automatically clear whether all individual programmers are willing to share their contributions permanently and irrevocably as free software. If some programmers are allowed to withdraw their contributions from the project at their will, it would cause great uncertainty to the project. For example, the legal uncertainty caused by the

patches assigned or solicited, although Torvalds is justly famous for encouraging people to work on particular problems, but only if they wanted to." Keltz, *Two Bits*, p.220

¹⁶² Benkler observes: "Collaborative production systems pose an information problem. The question that individual agents in such a system need to solve in order to be productive is what they should do. Markets solve this problem by attaching price signals to alternative courses of action. Firms solve this problem by assigning different signals from different agents different weights. To wit, what a manager says matters. In order to perform these functions, both markets and firms need to specify the object of the signal sufficiently so that property, contract, and managerial instructions can be used to differentiate between agents, efforts, resources, and potential combinations thereof." Benkler, "Coase's Penguin, or, Linux and 'The Nature of the Firm'" (2002) 112, (3) *Yale Law Journal* 369, at 375

¹⁶³ Benkler, *Wealth of Networks: How Social Production Transforms Markets and Freedom* (New Haven: Yale University Press, 2006), p. 414

withdrawal of Gosling’s code from “Emacs commune” in the pre-GPL era was real and tremendous.¹⁶⁴ So the GPL as an umbrella agreement reduces this uncertainty by standardising a few minimum legal commitments that are necessary to prevent a project from disintegrating, although it never presentiates any detailed obligations concerning substantive contributions. These minimum obligations mainly concern the availability of source code when a contribution is conveyed to the public. In other words, the GPL makes sure all publicly conveyed code, verbatim or modified, must be made available for the public to freely copy, use, modify and redistribute.¹⁶⁵ In this light, the GPL as an umbrella adds a level of certainty by making sure that it would provide a legal infrastructure where all unpresentiated peer-produced contributions can be legally compatible free software to stay in the same collaborative project.

In summary, the GPL as an umbrella agreement addresses the need for *flexibility* by not presentiating the substantive content of peer production in a continuing relation, and at the same time it addresses the need for *certainty* by standardising the minimum legal commitment to make these peer-produced contributions legally compatible. Again it is worth reemphasising that although the GPL is an important umbrella agreement that balances these two needs in a collaborative relation, it does not equal, but only facilitate, this whole relation. Just as Macneil reminds: “Under the relational approach, express terms in contracts are no more than an extremely important part of a dense web of relations.”¹⁶⁶ The next chapter will address the issue concerning FOSS authorship, which is another important part of the relational web but is too complex to be fully explained by the express text of the GPL.

4.5 Conclusion

This chapter examines some difficult issues concerning the FOSS licences as non-negotiated standard form contracts. From a strictly classical contractual view, most FOSS licensing schemes would lack affirmatively expressed consent from licensees

¹⁶⁴ See Section 2.3.2, Chapter 2

¹⁶⁵ See Section 3.5, Chapter 3

¹⁶⁶ Ian R. Macneil, “Reflection on Relational Contract Theory after a Neo-classical Seminar”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. By David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003) p.208

to make themselves binding. There are two alternatives to this classical approach. One is the neoclassical contract law as represented by the *ProCD* ruling, which reorients the legitimation of standard forms towards the goal of maximising individual utility gains. The other alternative is the Macneilian relational contract approach, which is endorsed by this chapter. I argue that the relational approach is more appropriate to deal with a relation-rich FOSS collaborative experience than the classical or neoclassical contract model. My examination of the GPL as a relational umbrella agreement shows its role in maintaining a balance between the need for *flexibility* in identifying the creative tasks by programmers themselves and need for legal *certainty* in producing legally compatible contributions to stay in one project irrevocably as free software. Based on this relational insight, the next chapter will explore the diverse motivational forces behind FOSS authorship in relation to FOSS licensing.

Chapter 5 The Idea of Authorship in FOSS Licensing

5.1 Introduction

Who are the “authors” of free and open source software? How do programmers claim their “authorship” in collaboratively created FOSS projects? To which extent does this FOSS “authorship” deviate from the eighteenth-century Romantic author vision that has purportedly shored up the modern copyright law¹? Do FOSS licensing schemes correspondingly carve out a unique legal persona for programmers working in collaboration that is detached from Romantic aesthetics? Compared with many scholarly writings on legal enforcement of FOSS licences including copyleft licences, the size of the legal literature tackling above questions about FOSS authorship is considerably small². Dusollier observes that “[t]he author is barely mentioned in copyleft, despite playing a prominent role in the system” and this marked absence “unfortunately conceals the importance of the author figure in the philosophical model of copyleft.”³ As all copyleft licences are copyright licences in the first place, Dusollier’s observation tallies with Ginsburg’s worry that “the figure of the author is too-often absent” in “contemporary debates over copyright” and this absence may only lead to an incomplete understanding of “copyright’s role in fostering creativity.”⁴ Similarly, the lack of discussion of authorship in FOSS licensing schemes can also risk losing sight of the whole picture of the role of FOSS licensing

¹ For a definitive account of the Romantic author vision and modern copyright, see Martha Woodmansee, “The Genius and Copyright” in *The Author, Art, and the Market—Reading the History of Aesthetics* (NY: Columbia University Press, 1994) originally published in (1984) 17 *Eighteenth-Century Studies* 425, titled “The Genius and Copyright: Economic and Legal Conditions of the Emergence of the ‘Author’” (Hereafter “The Genius and Copyright”)

See also James Boyle, *Shamans, Software, and Spleens—Law and the Construction of the Information Society*, (Cambridge, Mass.: Harvard University Press, 1996); for two important critiques of Boyle’s treatment of Romantic authorship and law, see Mark Lemley, “Romantic Authorship and the Rhetoric of Property”, (1997) 75 *Texas Law Review* 873 (hereafter “Romantic Authorship”) and Pamela Samuelson, “The Quest for Enabling Metaphors for Law and Lawyering in the Information Age” (1996) 94 (6) *Michigan Law Review* 2029 (Hereafter “Enabling Metaphors”)

² For example, Dusollier’s attempt to link FOSS authorship with postmodern aesthetics in a 2003 law journal article still remains arguably the most important contribution in the legal literature. See Severine Dusollier, “Open Source and Copyleft: Authorship Reconsidered?” (2003) 26 *Columbia Journal of Law and the Arts* 283

³ *ibid.*, p.288

⁴ Jane C. Ginsburg, “The Concept of Authorship in Comparative Copyright Law” (2003) 52 *DePaul Law Review* 1063 at 1063

in *coordinating* FOSS collaboration. As the last lap of my journey of exploring FOSS licensing, my task in this chapter is exactly to map out the complex idea of collaborative authorship as manifested by FOSS licensing schemes.

The main thrust of this chapter is that “authors” do exist in FOSS projects and they exist not only at the individual level, but more importantly, also at the collective (project) level. At the individual level, Dusollier suggests that authors are the “initiators” of each individually created piece of code,⁵ but she does not go further to elaborate in detail how these individual contributions are later integrated into a *collective* work, which can be attributed to a project as a whole.⁶ What is ignored here is exactly FOSS authorship at the collective level, which is an equally important but poorly understood matter. An “authorless” project at the collective level would simply be a failed project where individually created contributions do not aggregate into a coherent whole. An “authored” FOSS project is not content oriented towards producing a Babel of unrelated software fragments in a radically decentralised environment, but it also wants every contributor to *coordinate* with one another.⁷ It is exactly these coordinating efforts that give birth to the collective authorship, which can be held responsible and deserve credit for the production of an integrated FOSS

⁵ Dusollier seems to think that these individual authors then fully withdraw their authorial control over their creation once and for all under FOSS licences. In this scenario, the software becomes a kind of constantly reformulateable free-flowing postmodern “work”, whose link with its initial individual authors is irreversibly lost: “Once the work is made available to the public, the formerly unwavering link to the author becomes blurred. The author is no longer considered ‘the initiator of the collective work.’ Furthermore, the integrity of the work—that element which reflects authorial personality and justifies an extensive moral right in Continental doctrine—no longer means much. In this sense, the author resembles the figure of postmodern literary aesthetics of Foucault’s ‘founder of discursivity.’” As the initiator of an open discourse—of an ever-evolving work—the author of an element of a collective creation in copyleft finds her particular contribution diluted by the whole of successive contribution. The ‘work’ in the copyleft regime is software in constant (re)-formation; it is the production of meaning from different convergent or successive artistic practices.” Dusollier, *supra* note 2, pp. 294-5

⁶ Dusollier only mentions collective authorship in passing at the end of the concluding part of her essay. She seems to argue that the collective work is made possible with the help of copyleft, but she does not explain how the legal mechanism of copyleft exactly helps to coordinate individual authors’ interests in more detail: “The author is not only the initial founder of a discourse and instigator of a creation of which her contribution is only the first stage. She is also the figure by whom *the whole of the collective creation* finds itself marked by the stamp of freedom. In the chain of contributions, of works which will come to add incrementally to the first act, none will be able to escape the refusal of intellectual property rights exerted in a proprietary and exclusive manner. Foucault’s desire for greater cultural freedom is brought to life in copyleft.”(added emphasise) *ibid.*, at 295

⁷ See also Section 1.3.1, Chapter 1, where I argue that “collaboration” in any successful FOSS project has two defining aspects: it is not only 1) *radically decentralised* but also 2) *coordinated* among a large number of contributors. This chapter elaborates the second aspect of collaboration in terms of FOSS licences’ role in coordinating contributors’ legal commitment.

project. Of course, this *collective authorship* in the whole project should not be conflated with the *individual authorship* in each individual contribution. The former by no means compromises the latter, which is also respected within the FOSS community. A full evaluation of FOSS authorship in relation to FOSS licensing should be scrutinised at both collaborative and individual levels, though the existing literature does not tend to be discerning enough to differentiate the two. In particular, I will highlight the pivotal role of a small core group of lead authors—who are the lead programmers or coordinators in a FOSS project—to integrate individual authors’ creation into a collective work. I call them “stewards”, whose coordinating efforts make their authorship quite different from the conventional author-owners claiming exclusive rights under the intellectual property regime.⁸

The chapter is divided into three parts. In the first part (Section 5.2), I examine whether the Romantic genius vision fits with FOSS authorship at both individual and collective levels. At the individual level, there is no shortage of extremely talented FOSS programming “geniuses” in the community. However, the making of a collaborative FOSS project always goes beyond celebrating the virtuosity of these individual “geniuses”. *Individually* created contributions must be aggregated into a workable coherent whole, which can then be *collectively* held responsible and deserve credit for an integrated project as a whole. Most interestingly, individual author-geniuses do not simply disappear under the shadow the collective FOSS authorship, but a few most active and enthusiastic ones, who usually become project-leaders/coordinators, stand out as the *author-stewards* for certain projects for a sustaining period of time. I point out that, although these coordinators play a tremendously important role in channelling individual authorship into collaborative authorship, their author-stewardship is an understudied phenomenon by legal scholarship. In the second part (Section 5.3), I explore the FOSS programmers’ legal persona, which has developed to a large extent independently of the Romantic author vision. I try to demonstrate how FOSS programmers use their licensing schemes to claim their authorship at both individual and collective levels, despite the fact that

⁸ The alignment of authorship with stewardship in FOSS projects is in counterpoint to Mark Rose’s famous observation that modern authorship is distinguished by its link with ownership: “the author is conceived as the originator and therefore the owner of a special kind of commodity, the work.” Rose, *Authors and Owners—The Invention of Copyright*, (Cambridge, Mass. & London: Harvard University Press, 1993) p.1

Anglo-American law does not statutorily give software programmes a standalone attribution right. In particular, I will tackle the problem as to how project-leaders, in the capacity of author-stewards, enlist trademark law to protect the reputation or goodwill for their project as a whole. The third part (Section 5.4) concludes.

5.2 Individual and Collective “Authors” in FOSS Programming

This section discusses the authorial consciousness of FOSS programmers at both individual and collective levels. It examines the extent to which Romantic aesthetics is still viable in explaining the actual practice of FOSS programming. It shows that the individualistic Romantic author vision may still be applied to *individual* authorship of contributed code, but it is too inadequate to account for FOSS authorship at the *collective* level. In particular, the Romantic vision seems to suffer from a blind spot by failing to recognise project-leaders’ unique authorial role as “stewards”, who are instrumental in channelling individual authors’ efforts into one collective authorship that can be held responsible and deserve credit for a FOSS project as a whole.

5.2.1 Debating the Legacy of Romantic Aesthetics

The *individualised* “author”, who is credited as the sole origin of a creative work, is a construct of relatively recent pedigree. Woodmansee, in her 1984 essay, has provided a definitive account of the rise of the self-inspired “genius” and its repercussion in modern copyright law since the Western Romantic Movement beginning in the second part of the eighteenth century. Literary creators, equipped with the ammunition from Romantic aesthetics, lifted themselves out of the unimaginative rank of *craftsmen*, and they become author-geniuses capable of making totally original contributions derived from their unique creative personality. The oft-quoted Wordsworth’s testimony made in 1815 is an exemplary statement asserting the literary author-genius to be the sole fountain of his *original* creation:

Of genius the only proof is, the act of doing well what is worthy to be done, and what was never done before: Of genius in the fine arts, the only infallible sign is the widening the sphere of human sensibility, for the delight, honor, and benefit of human nature. *Genius is the introduction of a new element into the*

intellectual universe: or, if that be not allowed, it is the application of powers to objects on which they had not before been exercised, or the employment of them in such a manner as to produce effects hitherto unknown.⁹ (added emphasis)

The Romantic vision of author as a Wordsworthian genius capable of introducing “a new element into the intellectual universe” has arguably exerted an indelible and tremendous influence in shaping the contour of modern copyright law in the western world. Woodmansee points out that modern copyright is exactly built upon this cult of the author-genius:

Our laws of intellectual property are rooted in the century-long reconceptualization of the creative process which culminated in high Romantic pronouncements like Wordsworth’s to the effect that this process *ought* to be solitary, or individual, and introduce ‘a new element into the intellectual universe.’ Both Anglo-American ‘copyright’ and Continental ‘authors’ rights’ achieve their modern form in this critical ferment, and today a piece of writing or other creative product may claim legal protection only insofar as it is determined to be a unique, original product of the intellection of a unique individual (or identifiable individuals).¹⁰ (original emphasis)

In the more specific area of software copyright, there is no shortage of academic works that bear out Woodmansee’s worry about law’s uncritical acceptance of the Romantic mode of solitary and individualised authorship. Jaszi, an ardent champion of Woodmansee’s thesis, observes that “lawyers and judges have invoked the vision of the Romantic ‘author-genius’ in rationalizing the extension of copyright protection to computer software”, because software programs are “no less inspired than traditional literal works, and that the imaginative process of the programmer are analogous to those of the literary ‘author’.”¹¹ It is worth noting that the main source that Jaszi relies upon to make his observation is another influential article titled

⁹ William Wordsworth, “Essay, Supplementary to the Preface”, quoted in Woodmansee, “On the Author Effect: Recovering Collectivity” (1992) 10 *Cardozo Arts and Entertainment Law Journal* 279 at 280 (Hereafter “On the Author Effect”)

¹⁰ Martha Woodmansee, “On the Author Effect”, *ibid.*, at 291-2

¹¹ Peter Jaszi, “On the Author Effect: Contemporary Copyright and Collective Creativity”, (1992) 10 *Cardozo Arts and Entertainment Law Journal* 293 at 297-8

“Silicon Epics and Binary Bards” (hereafter “Silicon Epics”) written by Anthony Clapes and his colleagues in 1987¹² (three years after Woodmansee’s essay on “the Genius and Copyright” was first published). In “Silicon Epics”, Clapes *et. al.* straightforwardly liken software to “the arcane epic poetry of the Information Age”¹³ and a programmer is correspondingly the ‘poet’ of his poetic creation. It is emphasised that software should not be treated differently from literary works, because they are also “works of authorship in which the range and variety of expression are broad and deep” and software as “works of authorship exhibit all the attributes of literary works of a kind with which the general public and copyright laws are already quite conversant.”¹⁴ Note that the programmer-as-poet vision is not preached by “Silicon Epics” for the first time, but it comes from Frederick Brooks’ 1975 classical work on software design, which is often credited as an earlier source equating programmer with poet:

The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by *exertion of the imagination*. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures[...] ¹⁵ (added emphasis)

From Brooks’ point of view, programming is by no means a mindless job but it involves author’s “exertion of the imagination” upon his creation, and it is also enjoyable and fun. The programmer-author has the “sheer joy of making things” that is new and original: “As the child delights in his mud pie, so the adult enjoys building things, especially things of *his own design*. I think this delight [in programming] must be an image of God’s delight in making things, a delight shown in the distinctness of newness of each leaf and each snowflake.”¹⁶ The authors of “Silicon Epics” seem content to use Brooks’s author-as-god metaphor to defend the extension of US copyright law to software as previously recommended by the Mel

¹² Anthony Clapes, Patrick Lynch, and Mark R. Steinberg, “Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs” (1987) 34 *UCLA Law Review* 1493

¹³ *ibid.*, at 1584

¹⁴ *ibid.*

¹⁵ Brooks, *Mythical Man-Month*, pp.7-8, quoted by Clapes *et. al.*, “Silicon Epics”, *ibid.*, at 1497

¹⁶ *ibid.*, at 1496-7

Nimmer and National Commission on New Technological Uses of Copyrighted Works (CONTU).¹⁷ Note that this 1987 “Silicon Epics” essay is not a single isolated effort to invoke Romantic authorship to justify copyright protection for software, but it has a sustained appeal. For example, Miller, in a later *Harvard Law Review* article, makes a similar authorship argument that “imagination, originality, and creativity involved in writing a program is comparable to that involved in more time-honored literary works and far exceeds various mundane efforts that have long enjoyed protection under the copyright rubric”¹⁸. For this reason, he also reaches the conclusion that the unique creative expression from individual software programmers deserves copyright protection:

[...] the communicative precision required of a computer programmer is not unlike the discipline that a poet must achieve to convey a complex message within the confines of a tightly constrained meter or that of a composer who must work within the limited ranges of musical instruments or of the human voice. In each case, the copyright law rewards the *author's imagination and originality of expression* in the hope of encouraging further creative productivity.¹⁹ (added emphasis)

Both Clapes and Miller’s articles have vindicated Woodmansee’s observation that “creative product may claim legal protection only insofar as it is determined to be a unique, *original* product of the intellection of a unique *individual (or identifiable individuals)*”²⁰ (added emphasis). It is not difficult to find the Romantic creative mode, which is arguably the aesthetical mooring of modern copyright law, is based a presumption that the “original” contribution must come from a unique identifiable “individual”. I argue that this Romantic conception of “originality” and “individuality” fails to account for the complex phenomenon FOSS programming on two grounds. Firstly, the Romantic view of “originality” ignores the fact that

¹⁷ “It is this factual underpinning that was persuasive to Mel Nimmer and the majority of CONTU members in recommending only modest changes in the Copyright Act so that the full body of copyright law would apply to computer programs.” Clapes, “Silicon Epics”, at 1583; For CONTU’s report that analogises software to literary work, see also the discussion in Section 3.3, Chapter 3 of this dissertation.

¹⁸ Arthur R. Miller, “Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU” (1993) 106 *Harvard Law Review* 977 at 983-4

¹⁹ *ibid.*, at 984

²⁰ Woodmansee, “On the Author Effect”, *supra* note 9

programming is also an engineering discipline that has always prized intelligently reusing *old* elements in solving practical problems since its early hacker culture. Secondly, the Romantic conception of “individuality” (i.e., author as solitary and self-inspired genius) makes itself difficult to explain the practice of *collaborative* FOSS programming in a radically decentralised environment. I will explain both grounds in turn now.

5.2.2 Programming as an Engineering Discipline: Questioning “Originality”

The first ground calls into question the Romantic analogy of software programmers to Wordsworthian literary geniuses, whose unique “original” creative expression must introduce “a new element into the intellectual universe”. In fact, FOSS programming is not always about creating new things out of nothing in a rarefied atmosphere, but it is also an engineering discipline seeking to solve practical problems by using old and pre-existing technical solutions.²¹ Samuelson *et. al.* find Clapes or Miller’s attempt to treat programmers as literary author-geniuses ultimately fails²², but argue that a “well-designed program is thus akin to the work of a talented engineer whose skilled efforts in applying know-how, *accumulated from years of experience and training*, yields a successful design for a bridge or other useful product.”²³ There are two characteristics to this programmer-as-engineer view. First, programmers do not merely compose code as literary text, but more importantly they produce utilitarian artifacts that perform certain functions, from which the primary value of the program is derived.²⁴ Secondly, programming as an

²¹ The development of US software case law, especially after the 1992 *Altai* case where software was treated partially as a functional object and its non-expressive elements were accordingly excluded from copyright protection, seems to fly in the face of the Romantic argument that programmers are “original” literary writers. In other words, software copyright law after *Altai* seems not go down the exact trajectory as laid down by Romantic aesthetics, but it seems to veer onto a non-aesthetical course where programmers are recognised as “engineers”. See *Computer Associates International, Inc. v. Altai, Inc.*, 982 F.2d 693 (2d Cir.1992)

²² Pamela Samuelson, “Enabling Metaphors” *supra* note 1 at 2038-9; See also Lemley, “Romantic Authorship”, *supra* note 1 at 894

²³ Pamela Samuelson, Randall Davis, Mitchell D. Kapor, and J. H. Reichman, *A Manifesto Concerning the Legal Protection of Computer Programs*, (1994) 94 (8) *Columbia Law Review* 2308 at 2332 (hereafter *Manifesto*)

²⁴ “While conceiving of programs as texts is not incorrect, it is seriously incomplete. A crucially important characteristic of programs is that they behave; programs exist to make computers perform tasks. Program behavior consists of all the actions that a computer can perform by executing program instructions. [...] Behavior is not a secondary by-product of a program, but rather an essential part of what programs are. To put the point starkly: No one would want to buy a program that did not behave,

engineering discipline is not just about bringing “a new element into the intellectual universe”, but it is also about intelligently reusing and combining old elements that have been “accumulated from years of experience and training”. The second characteristic directly challenges the “originality” of Romantic authors and calls for recognising the incremental and accumulative mode of creativity in software engineering:

The products of software engineering almost invariably contain *admixtures of old and new elements*. Some consist almost entirely of old elements. The innovation in such programs may lie in the manner in which the known elements have been combined in a new and efficient manner. Or it may come from combining some new elements with well-known elements in order to achieve the same result in a new way. When we speak of programs as "industrial compilations of applied know-how," it is in recognition of the frequency with which software engineering involves *the reuse of known elements*. Use of skilled efforts to construct programs brings about *cumulative, incremental* innovation characteristic of engineering disciplines.²⁵ (added emphasis)

Samuelson’s depiction of software programmers as engineers, whose innovation is incremental and cumulative and involves skilled reusing of old elements, shows that software programmers’ authorial consciousness cannot be solely determined by Romantic *aesthetic* thoughts but there is also a strong *technical* dimension to the issue. The awareness that programmers are engineers or technicians is critical to understand the non-Romantic (incremental and cumulative) mode of practical creativity in collaborative FOSS programming. Weber observes: “Open source is first and foremost an engineering culture—bottom up, pragmatic, and grounded heavily in experience rather than theory.”²⁶ In fact, this engineering culture can be exactly stretched back to the early MIT-style hacker culture, where the ethos of sharing and reusing of existing solutions to technical problems was strong and robust

i.e., that did nothing, no matter how elegant the source code "prose" expressing that nothing.” *ibid.*, pp. 2316-7

²⁵ Samuelson *et. al. Manifesto*, supra note 23 at 2332

²⁶ Weber, *Success*, p.164

among computer hackers²⁷. For hackers and later FOSS programmers, it is unnecessarily wasteful to reinvent the wheel from scratch, however original this reinvention may be without copying from other sources. It is argued that there is “an ethical duty of hackers to share their expertise by writing open-source code and facilitating access to information and to computing resources wherever possible.”²⁸ However, the rise of private intellectual property in software and its accompanied pursuit for “originality” significantly diminished this sharing and reusing ethos since the late 1970s. It pushed the common activities of sharing and reusing under the hacker culture into a newly created category known as software “piracy” or “theft”, which was exactly the kind of “crime” that Stallman was accused of during his conflict with the proprietary software company Symbolics over the Lisp programming language initially co-developed at the MIT AI Lab.²⁹ In this sense, the hacker culture grounded in engineers’ practical intelligence is to a large extent a victim of software copyright law’s obsession with Romantic mode of “originality”, which refuses to see the more incremental and cumulative type of creativity.

5.2.3 Stewarding a FOSS Project: Questioning “Individuality”

The second ground questions authors’ “individuality” as assumed by Romantic aesthetics. As a general matter, this individualistic vision tends to attract two types of criticism. First, literary theorists blame it for neglecting the prevalent *collective* creative processes in the contemporary time.³⁰ Secondly, legal scholars are not satisfied with the lack of precise guide that Romantic aesthetics is able to provide for the actual development of legal doctrines of intellectual property. In particular, the Romantic individualistic vision is least competent in telling what law can do when there is a dispute between upstream and downstream authors. In this light, Lemley points out that Romantic authorship ultimately fails to inform how to balance the interests among what he calls “first and second generations” of authors:

²⁷ See Section 2.2.1, Chapter 2 of this dissertation for more detail.

²⁸ “The Hacker Ethic” in Jargon File, compiled by Raymond, at <http://www.catb.org/jargon/html/H/hacker-ethic.html>

²⁹ For more detail of Symbolics incident, see Section 2.3.2 Chapter 2

³⁰ For example, Woodmansee observes that “electronic technology is hastening the demise of the illusion that writing is solitary and originary” and the “writing” practice has become frequently collective in the electronic age. See Martha Woodmansee, “On the Author Effect: Recovering Collectivity” (1992) 10 *Cardozo Arts and Entertainment Law Journal* 279 at 289

The problem is that the idea of Romantic authorship does not necessarily lead one to favor one side or the other in a dispute between two types of authors—the first and second generations. One could invoke the language of romantic authorship either to demand strong copyright protection for a first-generation author or to demand an expansive interpretation of fair use for a second-generation author who has “transformed” a first-generation work.³¹

In the case of large-scale decentralised FOSS programming, this inter-generational authorial conflict can be exacerbated because there can be unlimited numbers of generations of programmers who work on the same piece of software. What complicates things further is that even founding members (i.e. the first generation of contributors) of a FOSS project do not always stay upstream in a project. They can rapidly move downstream when they use and modify contributions from later-generation programmers. FOSS licensing schemes to some extent pre-empt this problem by standardising all individual authors’ legal commitments when making a collaborative project.³² Having said this, I do not mean to give an impression that the collaborative efforts of a FOSS project can be reduced to its legal form as forged by these licensing schemes. Instead, I try to show that FOSS collaboration requires one lead programmer or a core team of programmers to make both legal and extra-legal arrangements to coordinate peer-produced contributions into a whole. I call these coordinators the “stewards” of FOSS projects. These stewards occupy a critically important “authorial” role in splicing individual authorial interests into a collective one.

Project-Leader/Coordinator as Author-Stewards

To understand FOSS project-leader/coordinators’ authorial persona as “steward” is not based on a false belief that FOSS programmers are incapable of making original contribution as *individuals*. There is no need to overcompensate for the weakness of Romantic aesthetics by denying the existence of programming “genius” in the FOSS community. In fact, many lead FOSS programmers are first known to the public for

³¹ See Lemley, “Romantic Authorship”, supra note 1 at 885

³² It is worth emphasising again the difference between copyleft and non-copyleft schemes. Copyleft licences makes *all* generations of authors have the same set of responsibility to commit their contributions to the commons, while non-copyleft licences schemes only limit this responsibility to the first generation of authors.

their virtuoso hacking skills rather than the relatively unsung role as project coordinators. Recall that Levy's 1984 hagiography of computer hackers, as its subtitle "Heroes of the Computer Revolution" suggests, is exactly a book full of larger-than-life programming geniuses in the pioneering days of hacking. One of the most notable of them is no doubt Richard Stallman, who is depicted by Levy much like a typical solitary Romantic genius. For example, during Stallman's personal struggle against the proprietary software company Symbolics which refused to share improvements of the Lisp language, Stallman's individual virtuosity in programming even won the admiration from a Symbolics employed programmer who commented: "[...] Stallman doesn't have anybody to argue with all night over there. He's working alone! It is *incredible* anyone could do this alone."³³ (original emphasis) Levy's writing is one of the earliest sources where Stallman gains this lonely (and sometimes unsociable) "genius" image. In 1990, this image is further reinforced by the prestigious MacArthur fellowship (also known as the "genius grant") given to Stallman literally in recognition of his "genius" status in the hacking world.³⁴ Nine years later, Michael Gross conducted an important interview further revealing that Stallman has exhibited all kinds of attributes normally associated with a solitary genius since his childhood as a lonely prodigy who was precociously talented and curious in many intellectual subjects.³⁵

However, Stallman's image as hacking genius sometimes overshadows his arguably more mundane administrative role as the coordinator-steward behind the GNU

³³ Levy, *Hackers*, p.426

³⁴ It is interesting to note that the very generous "genius grant" (\$240,000 in value including health insurance to Stallman) allows to Stallman to fully dedicate himself to the cause of free software without taking another full time job. Lerner explains what kind of people can be the awardees of the "genius grants": "The MacArthur fellowships, known as "genius grants," are awarded annually to exceptionally talented and creative people. This year's recipients include artists, human rights activists, mathematicians, and astronomers." Reuven M. Lerner, "Stallman wins \$ 240,000 in MacArthur Award", 18 July 1990 at <<http://tech.mit.edu/V110/N30/rms.30n.html>>

³⁵ In this interview, Stallman recalled that he was a very lonely child who had few friends, but he took an avid interest in many subjects: "I learned calculus when I was something like 7 or 8. So it wasn't hard for anyone to tell that I was interested in learning as much math and science as possible. For a couple of years when I was 14 to 16, I would go to the library and get two or three books a week about various subjects, like History, Math and Science. And I would read them all. At one point, I decided to learn Latin, so I got a first-year Latin textbook and went through it in a month, and then I got the second-year book and went through that in the next month". See Michael Gross, "Richard Stallman: High School Misfit, Symbol of Free Software, MacArthur-certified Genius", 2000, <<http://www.mgross.com/MoreThgsChng/interviews/stallman1.html>>; this interview about Stallman as a genius is important because it lays the foundation for a later book-length biography of Stallman by Sam Williams. Sam Williams, *Free as in Freedom—Richard Stallman's Crusade for Free Software*, O'Reilly 2002 at <<http://www.oreilly.com/openbook/freedom/>>

project. From the *whole* project's point of view, Stallman's persevering stewardship can be much more important than his personal geniushood. This is because Stallman's individual ingenuity, however great it is, would only be diluted in a robust project that can continue to attract a burgeoning number of contributors. At the same time, Stallman's role as coordinator would only be gradually accentuated over the time because a growing pool of programming talents needs more and more of his stewardship to channel their peer-produced contributions into a coherently integrated project. To put it succinctly, it is a lead programmer's tenacity rather than his ingenuity that gives him the stewardship, which matters most to the sustainability and longevity of a project.

The above point becomes even clearer when it is applied to the Linux project under the stewardship of Linus Torvalds. Raymond observes that Torvalds fails to be an individually "original" computing genius like Stallman in the first place, but he stands out as an engineer who is extremely good at implementing and integrating other people's contributions into the project. In this sense, Torvalds is considered to be a kind of lesser "genius of engineering and implementation", but he knows how to harness the collective intelligence from other hackers despite his lack of personal ingenuity:

But Linux didn't represent any awesome conceptual leap forward. *Linus is not (or at least, not yet) an innovative genius* of designing in the way that, say, Richard Stallman [... is]. Rather, Linus seems to me to be *a genius of engineering and implementation*, with a sixth sense for avoiding bugs and development dead-ends and a true knack for finding the minimum-effort path from point A to point B. Indeed, the whole design of Linux breathes this quality and mirrors Linus' essentially conservative and simplifying design approach.³⁶

It is not difficult to find that Raymond's argument has subtly widened the meaning of "genius", which has been conventionally pegged to the individualistic mode of "originality" under Romantic aesthetics. For Raymond, the "genius" may mean not only the self-inspired creator in the Romantic sense, but it may also include the more

³⁶ Eric Raymond, *The Cathedral and the Bazaar*, version 3.0 at <<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>> (hereafter *Cathedral*)

mundane but less individualistically “original” type of “genius of engineering and implementation” as epitomised by Torvalds who is good at using and reusing other people’s innovation. In fact, Torvalds is not alone for being this kind of lesser “genius”, but he belongs to a core stewardship team of subsystem maintainers for the Linux project. This team of maintainers are Linux’s “gatekeepers”, because they are responsible for reviewing all contributed patches, which can only be integrated into the mainline kernel with its approval.³⁷ Kelty illustrates how these gatekeepers do their daily job of reviewing and merging code submitted from other contributors:

Almost all of the decisions made by Torvalds and lieutenants were of a single kind: whether or not to incorporate a piece of code submitted by a volunteer. Each such decision was technically complex: insert the code, recompile the kernel, test to see if it works or if it produces any bugs, decide whether it is worth keeping, issue a new version with a log of the changes that were made. Although the various official leaders were given the authority to make such changes, coordination was still technically informal. Since they were all working on the same complex technical object, one person (Torvalds) ultimately needed to verify a final version, containing all the subparts, in order to make sure that it worked without breaking.³⁸

Furthermore, when a project-leader keeps coordinating or stewarding a project for a continuingly long time, he would not only be credited for his individual contribution, but more significantly, he would also get credit for his stewardship work that integrates other contributors’ efforts into a collective whole. To illustrate, Torvalds may claim two types of authorship for his work. On the one hand, he is the *individual author* of the code written by him; on the other hand, he is also the *stewardship*

³⁷ A 2009 Linux Foundation’s report shows briefly how patches are reviewed and approved by the subsystem maintainer term and it also shows that the code that Torvalds has merged under that release was slightly under 3%. “Patches do not normally pass directly into the mainline kernel; instead, they pass through one of one-hundred or so subsystem trees. Each subsystem tree is dedicated to a specific part of the kernel [...] and is under the control of the specific maintainer. When a subsystem maintainer accepts a patch into a subsystem tree, he or she will attach a “Sign-off-by” line to it. This line is a statement that the patch can be legally incorporated into the kernel; the sequence of signoff lines can be used to establish the path by which each change got into the kernel.” See Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update* at <<http://www.linuxfoundation.org/sites/main/files/publications/howwriteslinux.pdf>>, p.13

³⁸ Kelty, *Two Bits*, p.220

author who reviews, approves and integrates other people's contribution into the mainline Linux kernel. The former is familiar to the Romantic mode of individuated authorship, while the latter is a less familiar one but is crucial to the success of a large-scale collaborative FOSS project.

These two types of authorship may substantially overlap in a small budding project in its early formative stage, when a main programmer's individual contributions account for the most part of the program. At this stage, his significant individual authorship can easily give rise to project leadership, which is "essentially the same as ownership" as observed by Weber.³⁹ However, when the project scales up into a huge one, the lead programmer's individual authorship can be rapidly diluted to the extent that he can no longer justify his ownership/leadership of the whole program. Suppose that this programmer continues to be enthusiastic about leading the project ahead, the basis of his leadership practice must shift from the ever-dwindling *ownership* of the software to the ever-increasing *stewardship* responsibility in coordinating other people's contributions for the project.⁴⁰

This shift from ownership to stewardship is significant to a rounded understanding of project-leaders' unique authorial role in taking stewardship responsibility to forge collaboration in a FOSS project. Most importantly, countering the Romantic assumption of self-inspired authorship, FOSS leaders' author-stewardship seems to flesh out the two most important components of the author-as-steward thesis argued by Kwall. The first component comes from an awareness that an author himself is not the sole source of his own creation. Instead, *inspiration is externally endowed as a gift* that enables the author to make his own creation.⁴¹ In other words, the author is not entirely self-inspired, but he receives external inspiration as a gift that contains

³⁹ Weber, *Success*, p.166

⁴⁰ According to Lucy and Mitchell, the hallmark of stewardship is one's "responsibilities of *careful use*, rather than the extensive rights to exclude, control and alienate that are characteristic of private property" (added emphasis) and in a nutshell they are "duty-bearers" rather than "right-holders." William N.R. Lucy and Catherine Mitchell, "Replacing Private Property: The Case for Stewardship" (1996) 55 *Cambridge Law Journal* 566 at 584

⁴¹ Roberta Rosenthal Kwall, "The Author as Steward 'For Limited Times'", (2008) *Boston University Law Review* 685 at 703

Hyde vividly portrays how externally endowed inspiration work for creative artists: "We also rightly speak of intuition or inspiration as a gift. As the artist works, some portion of his creation is bestowed upon him. An idea pops into his head, a tune begins to play, a phrase comes to mind, a color falls in place on the canvas." Lewis Hyde, *The Gift: Imagination and the Erotic Life of Property* (New York: Vintage Books, 1983) p.xii

“unearned value”⁴² bestowed upon him. In a large-scale FOSS project, it is clear that every programmer benefits from other people’s contributions, and no one can claim to be the sole source of the whole program. Even for those founding members of projects, many of them try hard to avoid reinventing the wheel if there are existing technologies available for reuse. For example, Linus Torvalds did not start the Linux kernel from scratch in 1992, but his inspiration comes from the pedagogical Minix system initially developed by the Amsterdam-based computer scientist Andrew Tanenbaum in the late 1970s. Similarly, Stallman did not start the Emacs editor in the early 1980s from nothing, but the program was co-developed by a few programmers at the MIT Lab since the 1970s.

The second component of author-stewardship goes against rewarding creators with exclusive ownership *right*. Instead it evokes a sense of *responsibility* to offer an author’s work as a return gift back to the community where the author gets his externally endowed inspiration in the first place. Or to put it in Kwall’s words, this is the author’s stewardship responsibility to participate in “the cyclical dimension of creative enterprise”.⁴³ Lewis Hyde thinks that this responsibility actually comes from creators’ “labour of gratitude” which spurs creators to do something reciprocal for the external inspiration that is bestowed upon them early on.⁴⁴ In the history of FOSS development, Richard Stallman is exactly a programmer with a strong sense of stewardship responsibility to offer his software back to the community, though his view has lost much support facing the rise of commercial proprietary software. When Stallman started his GNU project in 1983 (two years before his first copyleft licence in 1985), his initial announcement of the project clearly indicates that he was driven by an ethical responsibility to share his software with the community: “I consider that the golden rule requires that if I like a program I must share it with other people

⁴² Schwarzenbach argues that a gift contains “unearned value” and the gift-receiver thus holds the gift with “unearned value” in stewardship. See Sibyl, Schwarzenbach, “Locke’s Two Conceptions of Property” (1988) 14 (2) *Social Theory and Practice* 141 at 146

⁴³ Roberta Rosenthal Kwall, “The Author as Steward ‘For Limited Times’”, (2008) *Boston University Law Review* 685 at 703

⁴⁴ “Between the time a gift comes to us and the time we pass it along, we suffer gratitude. Moreover, with gifts that are agents of change, it is only when the gift has worked in us, only when we have come up to its level, as it were, that we can give it away again. *Passing the gift along is the act of gratitude that finished the labor*. The transformation is not accomplished until we have the power to give the gift on our own terms. Therefore, the end of the labor of gratitude is similarity with the gift or with its donor.” (added emphasis) Lewis Hyde, *supra* note 41, p.47

who like it.”⁴⁵ His later experiment with copyleft, which makes programmers contribute (publicly released) modifications or improvements back to the community, further bears out his belief in “the cyclical dimension of creative enterprise” that is articulated in the legal language through software licensing.⁴⁶ To summarise, fusing the aforementioned two components together, author-stewardship manages to “blend[] an awareness of both externally endowed inspiration and the cyclical dimension of creative enterprise”⁴⁷, and it is very different from the conventional author-ownership model, which argues that the solitary self-inspired Romantic genius needs to be rewarded with private ownership to protect their creative works.

Reputational Incentive in the FOSS Community

It would be unrealistic to expect all FOSS programmers to harbour the same irresistibly strong and noble sense of stewardship responsibility to share software as Stallman does. FOSS licensing only prescribe a *minimum* set of stewardship responsibilities that secure software freedom, but they do not and cannot translate the entire MIT-style stewardship tradition in legal forms. In fact, individual programmers are often driven by a diversity of motivational forces ranging from a high sense of stewardship to highly self-interested motives. This is in line with Macneil’s general thesis (as discussed in Chapter 4) that participants to a long-term cooperative relation can be driven by a *spectrum* covering both individual utility-enhancement and non-utility-maximisation motives. Most interestingly, if Stallman’s high sense of stewardship represents the selfless non-utilitarian end of the spectrum, Eric Raymond’s argument in favour of reputational reward marks the individualistic utility end of the spectrum of FOSS programmers’ motivations. As I will soon show that, in the following Section 5.3, US case law (represented by the *Jacobsen* case) seems to be primarily based on a Raymondian individualistic understanding of FOSS

⁴⁵ Stallman further explains his ideal of sharing software with community: “By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.” Stallman, *GNU Initial Announcement*, 1983 at <<http://www.gnu.org/gnu/initial-announcement.html>>

⁴⁶ For the legal mechanism of copyleft in institutionalising the sharing norm, see for detail Section 3.5.1 Chapter 3

⁴⁷ Roberta Rosenthal Kwall, “The Author as Steward ‘For Limited Times’”, (2008) *Boston University Law Review* 685 at 703

programmers' motivation and leaves little room for a Stallmanian one, it is worthwhile explaining Raymond's approach in some detail now.

From a Raymondian point of view, it is futile to find whether it is morally right or wrong for programmers to share software with the community or "hoard" software privately.⁴⁸ It is more important to know that FOSS programmers are not fundamentally different from other self-interested rational human beings who seek to maximise their individual utility. A FOSS bazaar functions just like a free market and it is made of "a collection of *selfish agents attempting to maximize utility* which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved."⁴⁹ (added emphasis) However, what really makes FOSS bazaar unique is the fact that money is not primarily used as a measure of programmers' utility.⁵⁰ In this scenario, FOSS programmers use "reputational reward" as an alternative kind of utility that they intend to maximise. Or in Raymond's words, reputation simply has the "utility function" that satisfies FOSS programmers' egos.⁵¹ More specifically, there are three kinds of "utility" from reputation gains that may drive a programmer to participate in a FOSS project. Firstly, Raymond does not doubt that "good reputation among one's peers is a primary reward", i.e., it gives intrinsic satisfaction to a programmer. Secondly, one programmers' good reputation also tends "to attract attention and cooperation from others."⁵² In this sense, reputation is not a matter of individual motivation, but it also leads to collaboration. "If one is well known for generosity, intelligence, fair dealing, leadership ability, or other good qualities, it becomes much easier to persuade other people that they will gain by association with you."⁵³ Thirdly, "reputation may spill over and earn you higher status" in the world outside the FOSS

⁴⁸ "Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem." Raymond, *Cathedral*, supra note 36

⁴⁹ *ibid.*

⁵⁰ Raymond argues that "the open-source culture doesn't have anything much resembling money or an internal scarcity economy, so hackers cannot be pursuing anything very closely analogous to material wealth (e.g. the accumulation of scarcity tokens)." Raymond, Section 5, "Locke and Land Title" in *Homesteading the Noosphere*, 2002, at <<http://www.catb.org/~esr/writings/homesteading/homesteading/>> (hereafter *Noosphere*)

⁵¹ Raymond, *Cathedral*, supra note 36

⁵² Eric Raymond, *Noosphere*, supra note 50

⁵³ *ibid.*

community.⁵⁴ Note all the three reasons given by Raymond is not qualitatively different from lawyers' understanding of highly skilled individuals' attributional right to secure reputational gains as a kind of "human capital". For example, the legal scholar Catherine Fisk is one of the most notable champions of the view that the "reputation we develop for the work we do proves to the world the nature of our human capital."⁵⁵ In fact, reputation is believed to be a kind of "property" owned by individuals. "If professional reputation were property, it would be the most valuable property that most people own."⁵⁶ This reputational "property" is especially important to those highly skilled and highly educated workers, the value of whose work is otherwise difficult to be accurately assessed:

Particularly in the case of highly-educated or highly-skilled employees or people who possess a great deal of tacit knowledge, assessing the nature and value of human capital is difficult. The abilities of a software designer or music producer cannot be measured the way the speed of a typist or the competence of a machine operator can. When the cost of errors in assessment is great, or when assessments about human capital need to be made frequently or rapidly, easily, interpretable information *about* human capital is valuable because it reduces search costs. Thus, credit becomes a form of human capital itself because it translates and signals the existence of a deeper layer of human capital.⁵⁷

Although Raymond's reputation theory seems highly plausible to explain individual programmers' incentive to participate in FOSS programming, it suffers from at least two weaknesses that need to be addressed. First, Raymond's theory by no means gives the whole picture of multiple motivational forces behind FOSS programmers' efforts. An important empirical survey conducted by Lakhani and Wolf shows that reputation ranks rather low (11%) among all motivational forces that are most commonly recognised by programmers themselves. In particular, it shows that incentives such as programmers' intrinsic pleasure from FOSS programming for its own sake ("Code for project is intellectually stimulating to write", 44.9%; "Improve

⁵⁴ *ibid.*

⁵⁵ Catherine L. Fisk, "Credit Where It's Due: The law and Norms of Attribution" (2006) 95 *Georgetown Law Journal* 49 at 50

⁵⁶ *ibid.*, at 50

⁵⁷ *ibid.*, at 54

programming skills”, 41.3%)⁵⁸ and their desire to contribute software back to the community (“Believe that source code should be open”, 33.1%; “Feel personal obligation to contribute because use F/OSS”, 28.6%)⁵⁹ are regarded as more important than the reputational gains. (See Table 5.1) The second weakness of Raymond’s theory comes from its individualistic assumption. It does not really explain why programmers as selfish agents, who keen to maximise their individual reputational gain, would *collaborate* to create an integrated project. Weber finds that uncoordinated individual reputation competition might only introduce conflict among individual programmers, or even lead to disintegration of a project.⁶⁰ Furthermore, Raymond does not really delve into the important issue where FOSS programmers are also keen to protect the *collective* reputation of a whole project. For example, Stallman has campaigned very hard to make sure that the “GNU” project’s always get credit when it is used in juxtaposition with the Linux kernel.⁶¹ Most interestingly, FOSS project-leaders may not only resort to copyright to protect FOSS programmers’ attribution right (as the legal carrier of FOSS programmers’ reputation), but they may also evoke trademark law to protect a certain project’s name as the repository of the collective reputation or goodwill, which will be dealt with in the following section in some detail.

⁵⁸ These top two motivations (i.e. “Code for project is intellectually stimulating to write” and “Improve programming skills”) from the survey largely bear out Richard Sennett’s research on FOSS programmers’ obsession with the *quality* of their work for its own sake and their perfectionist tendency to improve their technical skills. (He uses Linux programmers as an example) See Richard Sennett, *Craftsman*, Richard Sennett, *The Craftsman* (New Haven & London: Yale University Press, 2008) pp.24-27

⁵⁹ These two obligations under the heading “obligation/community-based intrinsic motivations” seem to be very close to Lewis Hyde’s “labour of gratitude” argument as mentioned above in Stallman’s case, where Stallman feels obligated to share software with the community. See Hyde, *supra* note 41, p.47

⁶⁰ Weber lists many possibilities where programmers’ reputational incentives can fragment a project: “You might try to enhance your reputation by gravitating toward a project with the largest number of other programmers (because this choice increases the size of your audience, the number of people who would actively see your work). This would be inefficient on aggregate: Open source projects would then attract motivated people in proportion to their existing visibility and size, with a winner-take-all outcome. Or you might migrate toward projects that have the most difficult problems to solve, believing that you cannot make a reputation working on merely average problems (even for a big audience). But this choice would progressively raise the barriers to entry and make it difficult for new programmers to do anything valuable. Or you could engage in strategic forking—creating a new project for the purposes of becoming a leader and competing for the work of other programmers by distributing out the positive reputation returns more broadly within the community.” Weber, *Success*, pp.148-9

⁶¹ Stallman, “What’s in a Name?” at <<http://www.gnu.org/gnu/why-gnu-linux.html>>

Motivation		<i>Percentage of respondents indicating up to three statements that best reflect their reasons to contribute</i>	<i>Percentage of volunteer contributors</i>	<i>Percentage of paid contributors</i>
Enjoyment-based intrinsic motivation	Code for project is intellectually stimulating to write	44.9	46.1	43.1
Economic/extrinsic-based motivation	Improve programming skills	41.3	45.8	33.2
	Code needed for user need (Work need only)	33.8	19.3	55.7
	Code needed for user need (Nonwork need)	29.7	37.0	18.9
	Enhance professional status	17.5	13.9	22.8
Obligation/community-based intrinsic motivations	Believe that source code should be open	33.1	34.8	30.6
	Feel personal obligation to contribute because use F/OSS	28.6	29.6	26.9
	Like working with this development team	20.3	21.5	18.5
	Dislike proprietary software and want to defeat them	11.3	11.5	11.1
	Enhance reputation in F/OSS community	11.0	12.0	9.5

Note: This survey is also relevant to my argument about relational contract as discussed in Chapter 4. It shows that FOSS collaboration is not motivated solely by individuals' desire to maximise their material wealth, but it is driven by a diversity of values ranging from intrinsic satisfaction of code writing to reputational enhancement. This defeats the Easterbrookian assumption that all that licensees want is the lowest price, but it is in line with Macneil's viewpoint that participants are driven by both economic and non-economic motivational forces to collaborate under a long-term relational contract.

5.3 Development of the Legal Persona of FOSS Programmers

Although FOSS collaboration is largely based on the nonexclusive use and reuse of software components, this does not mean that FOSS programmers wish to give up the paternity right in their contributions. Instead, they are keen to claim credit where it is due and an efficient attribution system is necessary for that purpose. Almost all FOSS licences, regardless of being copyleft or non-copyleft, require downstream distributors to retain copyright notices including attribution information about the concerned projects and contributing programmers in all future public redistributions.

⁶² Reproduced from Karim R. Lakhani and Robert G. Wolf, "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects", in *Perspective on Free and Open Source Software*, eds. by Feller, Fitzgerald, Hissam & Lakhani (Cambridge, Mass.: MIT Press, 2005) pp.13-14

Legal scholars has been well aware that there is a strong norm of retaining correct attribution information in the FOSS community, where licences are used to make sure that credit as well as blame goes to the right projects and contributors. For example, Fisk observes that “[a]ttribution is important to many participants in the open source movement, even though exclusivity is shunned.”⁶³ She further points out that because FOSS programs are publicly *modifiable*, both upstream contributors and downstream modifiers should be correctly attributed for their respective contributions:

Open source licenses are an example of explicit effort to allocate credit and blame in attribution. All open source licenses seek to prevent bad *modifications* of the software from being attributed to the original authors. Although the explanation of the attribution requirements contained in the licenses are more focused on preventing wrongful attributions of blame than credit, presumably if a modification proves to be wonderful, the original authors will not get credit either.⁶⁴ (added emphasis)

In the same vein, the *Free Software Act* (FSA), which has been proposed by Free Software Consortium, nicely summarises the licensing norm of correct attribution in three points. “Authors’ rights shall be protected in the following way [...]: (a) The author of any free software program retains the right of attribution to his/her work. (b) Any modifier must acknowledge the authorship of the original program and the authorship of the modification. (c) All authorship must always be correctly attributed.”⁶⁵ This explanation and Fisk’s have something important in common. Both acknowledge the importance of a proper attribution system that should be even-handed on the authorial interests of both upstream authors and downstream modifying authors. It is worth noting that FOSS attribution is not merely about crediting these *individual* authors, but it is also about acknowledging the collective authorship that can be credited as the source of an integrated software artifact. Very often this integrated collective authorship is known to the general public through the

⁶³ Fisk, *supra* note 55 at 89

⁶⁴ *ibid.*, p.90

⁶⁵ Jaco Aizenman, Maureen O’Sullivan, Martin Pedersen, Pedro Rezende, Shilu Shah, Pia Smith, Jorge Villa, *Free Software Act (Draft)* (2004) 1 (4) *SCRIPT-ed* at <<http://www.law.ed.ac.uk/ahrc/script-ed/issue4/FS-Act.pdf>>

name of the corresponding project (e.g. “Linux”, “Apache”, “GNU”, etc.). Some projects would become brand names if they continue to provide products or services with certain a level of consistent quality for a sustaining period of time.

Although there is clearly a need to attribute FOSS authors at both individual and collective levels, Anglo-American legal system does not straightforwardly have a statutory attribution right for computer programmers. This has posed two difficult problems for the jurisprudence of FOSS licensing to tackle. First, how do FOSS programmers write their attribution requirement into the licensing condition of *copyright* licences? Secondly, how do FOSS programmers use the *trademark* as a proxy attribution right system to protect the reputation or goodwill of the collective authorship of the whole project? My analysis below finds that FOSS programmers, in order to compensate for the lack of statutory attribution right under copyright, have no choice but to assume the legal persona as the “owner” of intellectual property (either copyright or trademark, or both) of their contributions in the first place, which will then allow them to indirectly claim authorship legally. The situation is far from ideal and certain, but it partially works without changing the existing legislative structure about programmers’ attribution right. As copyright and trademark are different legal regimes in relation to authorial attribution, I will deal with “copyright” (in Section 5.3.1) and “trademark” (in Section 5.3.2) separately. The discussion will be followed by a further examination of how FOSS project leaders find the legal form of their stewardship in trademark in comparison with copyright (in Section 5.3.3).

5.3.1 Claiming FOSS Authorship under Law (I): Copyright

Strictly speaking, authors’ right to claim attribution of their creation, also known as the right of paternity, is not a proprietary right. Instead, it belongs to the “moral right” regime independent from a copyright owner’s economic right. Article *6bis* of the Berne Convention makes this clear: “Independently of the author’s economic

rights, and even after transfer of the said rights, the author shall have the right to claim authorship of the work [...].⁶⁶

Unfortunately, the Berne-type attribution right is not directly applicable to software programmers under Anglo-American copyright law. In the US, only visual artists but not computer programmers are entitled to the moral right of attribution.⁶⁷ In the UK, computer programmers are expressly excluded from having the right to be identified as author⁶⁸, and this attribution right is only conferred to a few non-programming creators who affirmatively assert their attributional interest.⁶⁹ However, the British copyright law traditionally gives authors a right against “false attribution”, which may still be applicable to computer programmers. This British indigenous moral right is not derived from the Berne Convention, but it harks back to the UK Fine Arts Copyright Act 1862, and has its reincarnations in respectively in s.43 of the Copyright Act 1956 and s.84 of CDPA 1988.⁷⁰ Lai finds this right against false attribution is an historical “anomaly” and it makes little sense for computer programmers to have it without having the right of attribution in the first place.⁷¹ In comparison, US programmers do not readily have a category against false attribution under their copyright law, but they may have an analogue protection under s.43(a) of

⁶⁶ The other moral right that is under same clause is known as authors’ right of integrity, which is the right to “object to any distortion, mutilation or other modification of, or other derogatory action in relation to, the said work, which would be prejudicial to his honour or reputation.” *Berne Convention the Berne Convention for the Protection of Literary and Artistic Works* (1971 revision with 1979 amendments)

However, some countries, such as the UK, have not adequately localised Berne’s moral rights regime into their national legislation. See Laddie *et. al. Modern Law of Copyright and Design*, p.586

⁶⁷ Visual Artists Right Act , 17 U.S.C s.106A ,

⁶⁸ S.79 (2) (a) CDPA

⁶⁹ s. 77. However, even for those non-programming creators, s.77(1) ends with a proviso saying that the right of attribution “is not infringed unless it has been asserted”. This requirement of assertion makes the CDPA out of line with Berne Convention no-formality requirement. Ginsburg believes that the CDPA’s text is a mistranslation of 6bis of Berne’s Convention, as “the drafters of the CDPA fashioned an obligation to assert authorship before the right to be recognized can take effect.” Jane C. Ginsburg, “The Right to Claim Authorship in U.S. Copyright and Trademarks Law”, (2004) 41 (2) *Houston Law Review* 263, p.291

⁷⁰ For a brief statutory history of this right, see Laddie *et. al. supra* note 66, pp.585-6

⁷¹ “If it is important to the author of a computer program not to have his work falsely attributed, it is difficulty to see why it is not important for him to be attributed as the author in the first place.” Lai, *The Copyright Protection of Computer Software in the United Kingdom* (Oxford and Portland, Oregon: Hart Publishing, 2000), p.20

the Lanham Act that codifies the common law action of passing off, which I will come back to in Section 5.3.3 in some detail.⁷²

Although Anglo-American copyright has largely failed to reproduce a Berne-type attribution regime to protect their programmers, this *lacuna* may be filled by private licensing schemes made by programmers in their capacity of *copyright owners*. This means these copyright licences make attribution ride on the proprietary right owned by FOSS developers. For this reason, Lastowka argues that Anglo-American copyright only protects attribution half-heartedly “in a collateral fashion”, where the attribution requirement needs to be “contracted in”:

It might be argued that copyright protects attribution in a collateral fashion .By *protecting works of creative authorship as property*, copyright enables the contractual protection of attribution. If an author can control the dissemination and reproduction of her work pursuant to copyright law, copyright law will grant her the contractual leverage to protect her attribution interests. (added emphasis)⁷³

So FOSS developers, in order to have their moral right of attribution enforceable under law, must take on the legal persona first as the *copyright owners*. The possibility of collateral protection of attribution via a copyright licence has been subject to a 2008 landmark ruling made by the US Court of Appeals for the Federal Circuit (CAFC) in *Jacobsen v. Katzer*, where the FOSS code in dispute was reproduced, modified and distributed without attributing to the original FOSS contributors.⁷⁴

“Collateral” Protection of Attribution in *Jacobsen v. Katzer*

⁷² See, for example, *Gilliam v ABC* 538 F.2d 14 (2d Cir.1976); *Follett v New American Library* 497 F. Supp. 304 (SDNY, 1980)

⁷³ Lastowka, “The Trademark Function of Authorship”, (2005) 85 *Boston University Law Review* 1172 at 1214 (here after “Trademark Function”)

Although I follow Lastowka to use the term “collateral” protection of attribution under copyright, my following discussion of the *Jacobsen* case shows that FOSS licences are not necessarily “contractual”, but they can be conditional licences where attribution is made the pre-condition to use copyrighted FOSS programs.

⁷⁴ 535 F.3d 1373 (Fed. Cir. 2008)

The *Jacobsen* case concerns a dispute over a FOSS project known as “Java Model Railroad Interface” (JMRI) that develops software controlling model railroads. JMRI is led by Professor Robert Jacobsen, who is a Berkeley physicist by profession and a model train hobbyist in his spare time. The JMRI code under dispute was then released under Artistic License (AL)1.0.⁷⁵ It is generally believed that this licence has explicitly created a private regime of moral rights enabling JMRI developers to have wider authorial control than allowed under the statutory language of the US copyright law. The Preamble of AL1.0 makes no effort to conceal this intent: “The intent of [AL] is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of *artistic control* over the development of the package [...]”.⁷⁶ (added emphasis) Fabricius comments that “the essential novelty” of AL lies exactly in its “granting the author more attribution and creative control than would be granted in the ordinary case of a copyright license to copy, distribute, and prepare derivative works”.⁷⁷ In this way, JMRI programmers are given “a private moral right” that is akin to the Section 106A of the U.S. Visual Artists Right Act providing attribution right only to certain visual artists.⁷⁸

The actual dispute revolves around a program called DecoderPro®, which is a sub-project of the JMRI.⁷⁹ In September 2006, the JMRI developers discovered that Matthew Katzer had copied and modified some DecoderPro files into his own

⁷⁵ JMRI now changes their licence to the GPL 2.0.

AL1.0 is not drafted by a lawyer but it is written by Larry Wall, a linguist by training and a reputable hacker who has invented the widely used open source Perl programming language. Wall, though not a lawyer, was convinced that copyright law was crucial to any open source project. A short extract below from Wall’s writing reflects his awareness of the importance of copyright: “A circle with a ‘c’ in it [i.e. ©]. Open Source lives or dies on copyright law. Our fond hope is that it lives. Please, let’s all do our part to keep it that way. If you have a chance to plug copyrights over patents, please do so. I know many of you are already plugging copyright over trade secrets. Let’s also uphold copyright law by respecting the wishes copyright holders, whether or not they are spelled out to the satisfaction of everyone’s lawyer.” See Larry Wall, “Diligence, Patience, and Humility”, in *Open Sources—Voices from the Opens Revolution*, DiBona, Ockman & Stone (eds.), (Sebastopol: O’Reilly, 1999) p.142

⁷⁶ Preamble, Artistic License 1.0

⁷⁷ Erich M. Fabricius, “Jacobsen v. Katzer: Failure of the Artistic License and Repercussions for Open Source” (2008) North Carolina Journal of Law & Technology 65, at 85

⁷⁸ *ibid.*, at 85

⁷⁹ “DecoderPro is able to easily configure more than 300 types of devices because hobbyists have contributed more than 100 decoder definition files. These definitions, produced by lots of separate contributors, are what makes the program so useful, since they express a model railroader’s view of how best to configure a particular device. DecoderPro first started using this approach in September 2001” JMRI, “JMRI Defense: Our Story So Far”, at <<http://www.decoderpro.com/k/History.shtml>>, last retrieved on 15 April 2010

proprietary product. At the same time Katzer deliberately removed the following information that would have identified JMRI contributors as authors of their code:

- 1) the authors' names,
- 2) JMRI copyright notices
- 3) references to the COPYING file
- 4) and identification of SourceForge or JMRI as the original source of the definition files, and
- 5) a description of how the files or computer code had been changed from the original source code.⁸⁰

Katzer did not dispute that his act of copying, but he contended that AL as a public licence had permitted him to copy the code and non-attribution of JMRI authors was not a cause of action itself under the US copyright law.⁸¹ So the difficult question is whether Katzer's act of deleting attribution information would lead to the infringement of the copyright of DecoderPro software.⁸² The trial court (District Court for the Northern District of California) took the view that the attribution requirement was merely a *contractual* covenant, the breach of which would only entitle JMRI developers to contractual damages but not an injunctive relief: "The condition that the user insert a prominent notice of attribution does not limit the scope of the license. Rather, Defendants' alleged violation of the conditions of the

⁸⁰ 535 F.3d 1373 at 1376

To illustrate, a typical JMRI file contain a piece of XML code showing who the author is. Here is an example from JMRI's webpage about the dispute:

```
<version author="Phil Grainger (phil.grainger@ca.com)"  
version="1" lastUpdated="20030805" />
```

The above XML code identifies three items of author-related information: 1) the author's name is "Phil Grainger" followed by his email address; 2) the version number is "1"; 3) it was last updated by the author on the date of 5 August 2003. However, when Katzer copied of JMRI files, he only retains the last two items, but he deliberately left out the first item about JMRI authors. JMRI developers further observes: "Original JMRI definition files contain the version, the date modified, the author's name, and a copyright notice. These have free-form content, so there are many formats. The version strings and the modification date strings in the KAM files are EXACTLY the same as those in the original JMRI files they were copied from. The author's name, however, was not copied into the KAM file, nor was the JMRI copyright information." JMRI, "JMRI Defense: Our Story So Far", at <<http://www.decoderpro.com/k/History.shtml>>; For more evidential information, see JMRI, "Copying Evidence: JMRI Defense: Evidence KAM Copied From JMRI", at <<http://jmri.sourceforge.net/k/copycomparison.shtml>>

⁸¹ The CAFC finds that the parties "do not dispute that Jacobsen is the holder of a copyright for certain materials distributed through his website. Katzer/Kamind also admits that portions of the DecoderPro software were copied, modified, and distributed as part of the Decoder Commander software. Accordingly, Jacobsen has made out a prima facie case of copyright infringement. Katzer/Kamind argues that they cannot be liable for copyright infringement because they had a license to use the material." 535 F.3d 1373 at 1379

⁸² It is noticed that the "heart of the argument on appeal concern whether the terms of the AL are conditions of, or merely covenants to, the copyright license." *Ibid.*, at 1380

license may have constituted a breach of the nonexclusive license, but does not create liability for copyright infringement where it would not otherwise exist.”⁸³

Failing to get an injunction, Jacobsen appealed the case to the CAFC, which reversed the district court ruling by arguing that attribution of JMRI developers is a necessary *condition* for the public to use their copyright material.⁸⁴ The failure to fulfil this condition would lead to infringement of copyright, which may give rise to the remedy of injunctive relief. Note that the CAFC does not straightforwardly enforce JMRI authors’ attribution *as such*: “Open source licensing restrictions are easily distinguished from mere ‘author attribution’ cases. Copyright law does not automatically protect the rights of authors to credit for copyrighted materials.”⁸⁵ Instead, FOSS developers’ attribution interest is only collaterally protected when they are the condition of a copyright licence that is intended to fulfil certain *economic* goals:

The clear language of the Artistic License creates conditions to protect the *economic rights* at issue in the granting of a public license. These conditions govern the rights to modify and distribute the computer programs and files included in the downloadable software package. The attribution and modification transparency requirements directly serve to drive traffic to the open source incubation page and to inform downstream users of the project, which is a significant *economic* goal of the copyright holder that the law will enforce.⁸⁶ (added emphasis)

This interpretation seems largely, if not entirely, to vindicate Lastowka’s view that copyright only gives collateral protection to author’s attribution.⁸⁷ In other words, the

⁸³ The District Court’s decision was quoted by CAFC, *ibid*.

⁸⁴ The appellate court made two observations to support its argument. First, AL states “on its face” that it creates “conditions”: “The intent of this document is to state the *conditions* under which a Package may be copied [...]” (added emphasis). Secondly, the US case law shows that the phrase “provided that” is typically employed to indicate that a certain *condition* has to be met. For example, the clause that Katzer was alleged to breach is Section 3 of AL stipulating that licensees, among other conditions, “may otherwise modify [their] copy of this Package in any way, *provided that* [they] insert a prominent notice in each changed file stating how and when [they] changed that file [...]” (added emphasis). AL 1.0.

⁸⁵ 535 F.3d 1373, FN5 at 1382

⁸⁶ 535 F.3d 1373 at 1382

⁸⁷ Lastowka argues that copyright give collateral protection of “creative authorship as property” through *contractual* arrangements. See Lastowka, “Trademark Function”, *supra* note 73

CAFC ruling interprets that FOSS authors need to claim their attributional interest as a matter of licensing condition in furtherance of the economic goal of copyright and thus wear the legal persona as copyright owner in the first place. The licensing “conditions” including the attribution requirement help to get upstream authors as “copyright holders” of their contributions always credited in downstream distributions:

The conditions set forth in the Artistic License are vital to enable the *copyright holder* to retain the ability to benefit from the work of downstream users. By requiring that users who modify or distribute the copyrighted material retain the reference to the original source files, downstream users are directed to Jacobsen's website. Thus, downstream users know about the collaborative effort to improve and expand the SourceForge project once they learn of the “upstream” project from a “downstream” distribution, and they may join in that effort.⁸⁸ (added emphasis)

Although the *Jacobsen* ruling is widely welcomed among FOSS developers and supporters of their cause⁸⁹, it is not entirely free from problems. There are at least two problems that are worth further scrutiny. The first one concerns an unintended consequence that FOSS author-owners may freely (mis)use *Jacobsen*-like licensing conditions to expand control over their works. It has been worried that *Jacobsen* can be an open-source version of the unpopular Seventh Circuit's *ProCD* decision⁹⁰. Narodick points out that *Jacobsen* “does not represent a fundamental shift in judicial policy, but the Federal Circuit's rationale in *Jacobsen* may justly concern the very open source software engineers who want to produce more programs in the future. This expansion of intellectual property rights effectively stacks the deck in favor of any software producer already in the market.”⁹¹ If Narodick is right, *Jacobsen* is just

⁸⁸ 535 F.3d 1373 (Fed. Cir. 2008) at 1381

⁸⁹ For example, Lessig wrote a blog entry celebrating the *Jacobsen* decision: “So for non-lawgeeks, [the *Jacobsen* case] won't seem important. But trust me, this is huge.”, see Lessig, “Huge and Important News: Free Licenses Upheld”, 13 August 13 2008, at <http://lessig.org/blog/2008/08/huge_and_important_news_free_1.html>

⁹⁰ For the discussion of *ProCD* and its influence in software licensing jurisprudence, see Chapter 4

⁹¹ It is observed that “*ProCD* has been gradually accepted by the federal judiciary up through and including the decision in *Jacobsen*.” Benjamin I. Narodick, “Smothered by Judicial Love: How *Jacobsen v. Katzer* Could Bring Open Source Software Development to a Standstill” (2010) 16 *Boston University Journal of Science and Technology Law* 264 at 279-281

another case where copyright owners expand their proprietary right through a private licensing scheme regardless of the software being FOSS or proprietary: in this case, the expansion yielded an author's attribution right; in another case, it could diminish or eliminate a user exception or effectively (as in *ProCD*) extend copyright protection to non-copyrightable materials. Gomulkiewicz is aware of a view that not all restrictions in a software licence will qualify as a non-contractual licensing "condition", but they should be limited to certain acts that directly concern or "touch upon" copyright. In this scenario, in order to decide whether a clause is too expansive to qualify as a copyright "condition" proper, questions must be asked how far or remotely this condition "touches on", or is related to, the right to copy, distribute or make derivative works under copyright. "The farther a purported condition strays from touching on an exclusive copyright, the less compelling the case that a licensee infringed a copyright by failing to abide by the condition."⁹² Unfortunately, in the case of moral rights, Gomulkiewicz himself is not sure whether attribution "touches upon" copyright, because it is an extremely uncertain "gray area" that is not related "directly to copying, distribution, or derivative works".⁹³

This uncertainty leads to a second problem: if CAFC's interpretation gives an impression of AL being a device to privately organise copyright holders' "economic" interests, does it really fall under the Easterbrookian neoclassical agenda to establish economic efficiency as the best justification for standard-form software licences? This is not entirely clear. CAFC's underlying philosophy seems to be only slightly more eclectic than the economic reductionist approach apparent in the *ProCD* rationale, which assumes that all software consumers or end-users need is the cheapest price with transaction cost saved from the unbargained standard-form

⁹² Gomulkiewicz gives some sample criteria about "touching upon" copyrights: "To qualify as a condition on the right to copy, for instance, the condition should relate to issues such as: Copying onto what? Using what to make copies? How many copies? What type of copies? Who can make copies? For a condition on the right to distribute, the condition should relate to issues such as: Where (and where not)? When? To whom? By whom? For how long? For a condition on the right to make derivative works, the condition should relate to: What type of works? Who can make derivatives? Analytically, this approach seems to make sense—copyright violations triggered by breach of a license condition should actually invoke copyrights." Robert Gomulkiewicz, "Conditions and Covenants in License Contracts: Tales from a Test of the Artistic License" (2009) 17 *Texas IP Law Journal* 335 at 354

⁹³ He also feels uncertain about the situation of copyleft. For example, GPL's share-alike provision "might not qualify" as a "condition" in this sense either, because it does not directly "touch upon" copyrights. *ibid.*, 355

licence.⁹⁴ The *Jacobsen* court is aware that FOSS is different from other traditional commercial copyright materials that are mainly sold for money, but the lack of monetary exchange with FOSS does not mean it cannot bring economic benefit to the FOSS author as the copyright holder:

Traditionally, copyright owners sold their copyrighted material in exchange for money. The lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration, however. There are substantial benefits, including economic benefits, to the creation and distribution of copyrighted works under public licenses that range far beyond traditional license royalties. For example, program creators may generate market share for their programs by providing certain components free of charge. *Similarly, a programmer or company may increase its national or international reputation by incubating open source projects. Improvement to a product can come rapidly and free of charge from an expert not even known to the copyright holder.*⁹⁵

The above paragraph shows that CAFC has noticed at least two types of non-monetary “economic” gains, the first being market share growth a certain FOSS product and the second being FOSS programmers’ boosted reputation that may attract a diversity of expertise needed to improve software itself. Note the second benefit from a good reputation (as italicised above in the CAFC’s decision) is not qualitatively dissimilar from a Raymondian economic understanding of the reputational incentive in FOSS production, where good reputation will attract cooperation from other experts: FOSS programmers’ “prestige is a good way [...] to attract attention and cooperation from others. If one is well known for generosity, intelligence, fair dealing, leadership ability, or other good qualities, it becomes much easier to persuade other people that they will gain by association with you.”⁹⁶ This is exactly a kind of non-monetary benefit that proprietary software owners would not have, because their users are merely consumers who are not allowed to modify or

⁹⁴ For ProCD’s economic reductionist approach, see Deborah Post, “Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook”, (2000) 16 *Touro Law Review* 1205; see also Section 4.4.1 Chapter 4 of this dissertation for more detail.

⁹⁵ 535 F.3d 1373, 1379

⁹⁶ Eric Raymond, *Noosphere*, supra note 50

improve the proprietary products. In this sense, the *Jacobsen* court seems to agree that reputation is a special human capital, which resembles but is not exactly the same as money. This view has been articulated by Rishab Ghosh who likens FOSS programmers' reputation to "a currency, i.e. a proxy, which greases the wheels of the economy", but it is subtly different from the monetary currency:

Unlike money, reputation is not fixed, nor does it come in the form of single numerical values. It may not even be cardinal. Moreover, while a monetary value in the form of price is the result of matching demand and supply over time, reputation is more hazy. In the common English sense, it is equivalent to price, having come about through the combination of multiple personal attestations (the equivalent of single money transactions).⁹⁷

Apart from being a non-monetary "hazy" currency, reputation also functions as a proxy-measure of quality of one's work as assessed by peer programmers. Weber finds that a FOSS "author is too close to the work and needs external measures of quality in order to know whether the work is good and how to improve it"⁹⁸ and that's why external assessment of reputation by peers is needed:

As is true of many technical and artistic disciplines, the quality of a programmer's mind and work is not easy for others to judge in standardized metrics. To know what is really good code and thus to assess the talent of a particular programmer takes a reasonable investment of time. The best programmers, then, have a clear incentive to reduce the energy that it takes for others to see and understand just how good they are. [...] The programmer participates in an open source project as a demonstrative act to show the quality of her work. Reputation within a well-informed and self-critical community becomes the most efficient proxy measure for that quality.⁹⁹

⁹⁷ Rishab Ayer Ghosh, "Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet" (1998) 3 (3) *First Monday* at <<http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/580/501>>

⁹⁸ Weber, *Success*, p.141

⁹⁹ Weber, *Success*, p.142

Fisk is also aware that reputation as a quality measure is not just in software area. In most cases of "highly-educated or highly-skilled employees or people who possess a great deal of tacit knowledge, assessing the nature and value of human capital is difficult": "The abilities of a software designer or

In short, the double function of reputation as both an incentive and a proxy-measure of quality largely bears out CAFC's analysis that the "lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration"¹⁰⁰, but the licensing condition requiring attribution can help to fulfil copyright owners' "economic" goal of organising FOSS production and circulation.

However, the *Jacobsen* court's economic approach, which makes attribution ride on the economic interest of the copyright holder, has to pay a price. It unfortunately screens out some "softer" non-economic values (e.g. "software freedom") that a true Macneilian relational licence would embody.¹⁰¹ For example, Stallman has insisted that the licensing conditions of the GPL are intended to advance "software freedom", which is an intrinsic value independent of programmers' economic interests.¹⁰² He further argues that accurate attribution to the names of "free software" projects is absolutely necessary to spread the ideal of "software freedom" behind their efforts. For example, as the "GNU" software is a significant integral component in the whole Linux operating system, Stallman is afraid that a common non-attribution to "GNU" would only leave people oblivious to the "software freedom" value that GNU programmers are keen to spread. For this reason, he is emphatic that the right way of attribution of the operating system is "GNU/Linux" rather than "Linux". He is even prompted to write an essay—"What's in a Name"—to stress the importance of attribution to GNU and the underpinning software freedom ideal, without which their efforts may be gradually watered down by the encroachment of proprietary software:

music producer cannot be measured the way the speed of a typist or the competence of a machine operator can. When the cost of errors in assessment is great, or when assessments about human capital need to be made frequently or rapidly, easily, interpretable information *about* human capital is valuable because it reduces search costs. Thus, credit becomes a form of human capital itself because it translates and signals the existence of a deeper layer of human capital." Fisk, *supra* note 55, p.54

¹⁰⁰ 535 F.3d 1373, 1379; *supra* note 95

¹⁰¹ Recall that, in Chapter 4, I argue that a Macneilian relational licence is very different from a neoclassical contract (as exemplified by Easterbrook's *ProCD* ruling) but it embodies a multiplicity of values including those non-economic values. See also William Whitford, "Ian Macneil's Contribution to Contracts Scholarship", (1985) *Wisconsin Law Review* 545

¹⁰² The *Jacobsen* court seems to be skewed towards the business-friendly "open source" approach advocated by Raymond, in order to argue that FOSS licensing conditions have the effect of furthering the economic goal of copyright holders. However, it seems difficult to square the more purist "free software" values with the *Jacobsen* ruling. Stallman argues that "free software is an ethical imperative, because only free software respects the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software "better"—in a practical sense only." See Stallman, "Why Open Source Misses the Point of Free Software" at <<http://www.gnu.org/philosophy/open-source-misses-the-point.html>>

Names convey meanings; our choice of names determines the meaning of what we say. An inappropriate name gives people the wrong idea. [...] Is it important whether people know the system's origin, history, and purpose? Yes—because people who forget history are often condemned to repeat it. The Free World that has developed around GNU/Linux is not guaranteed to survive; the problems that led us to develop GNU are not completely eradicated, and they threaten to come back.¹⁰³

According to Stallman, the GNU project desperately needs credit because their cause of fighting for software freedom is far from successful and the correct attribution to the project would constantly remind people of this cause: “If ‘the job’ [of the free software movement] really were done, if there were nothing at stake except credit, perhaps it would be wiser to let the matter drop. But we are not in that position. To inspire people to do the work that needs to be done, we need to be recognized for what we have already done. Please help us, by calling the operating system GNU/Linux.”¹⁰⁴ Interestingly, the *Jacobsen* ruling did mention the (GNU/) “Linux” system twice in passing, referring to “GNU” on the first occasion¹⁰⁵ but not the second one¹⁰⁶. It is worth noting that on the second occasion, the “Linux” system (without attributing to GNU) is quoted as an example to show why FOSS licensing is essential to fulfil the economic goal of “creative collaborative projects” in a most efficient way:

Open Source software projects invite computer programmers from around the world to view software code and make changes and improvements to it. Through such collaboration, software programs can often be *written and debugged faster and at lower cost* than if the copyright holder were required to do all of the work independently. In exchange and in consideration for this

¹⁰³ Stallman, “What’s in a Name?” at <<http://www.gnu.org/gnu/why-gnu-linux.html>>

¹⁰⁴ *ibid.*

¹⁰⁵ “Open source licensing has become a widely used method of creative collaboration that serves to advance the arts and sciences in a manner and at a pace that few could have imagined just a few decades ago. [...] Other public licenses support the *GNU/Linux* operating system [...]” 535 F.3d 1373 at 1378 (added emphasis)

¹⁰⁶ This time the court only mentions the “GNU” in juxtaposition with the “GPL”, but not with “Linux”, i.e., it does not use the term “GNU/Linux” as insisted by Stallman. The court writes that “the GNU General Public License, which is used for the Linux operating system, prohibits downstream users from charging for a license to the software.” *ibid.*, FN 2 at 1379

collaborative work, the copyright holder permits users to copy, modify and distribute the software code subject to conditions that serve to protect downstream users and to keep the code accessible.¹⁰⁷

Note that CAFC interprets FOSS licensing exactly as a cost-efficient way of manufacturing and improving software thanks to the collaborative intelligence that can be attracted to the job: because “software programs can often be *written and debugged faster and at lower cost* than if the copyright holder were required to do all of the work independently,” this is certainly in the economic interests of the copyright holders. “Software freedom” is not mentioned as one of most important non-economic values behind the GNU/Linux.¹⁰⁸ In this light, I think that the *Jacobsen* ruling, though hailed as a long waited victory for FOSS authors, does not go far enough to embrace the Macneilian message to respect non-economic values in building a relational contract. It merely substantiates Lastowka’s worry about the persistent influence of “utilitarian and property-centric view of copyright”¹⁰⁹ in software licensing jurisprudence. It also illuminates how the programmer’s attributional interest is “collaterally” protected under their legal persona as a utility-maximising property “owner”, who uses licensing conditions to advance their economic interest.¹¹⁰ Having said that, the situation is not entirely CAFC’s fault, because the court is restrained by the existing legislative framework where there is no stand-alone attribution right.

5.3.2 Claiming FOSS Authorship under Law (II): Trademark

Apart from relying on copyright for collateral protection of attribution, FOSS projects nowadays are also actively seeking trademark protection of their attributional interests. This is because trademarks designating the origin of goods or

¹⁰⁷ The paragraph is from the main text accompanying the FN2 of the ruling. *ibid.*

¹⁰⁸ Here my observation is not a critique of the CAFC’s failure to mention “software freedom”, because the court is constrained by previous case law and the copyright legislation where attribution on its own is not recognised. What I want do is only draw a Macneilian perspective that may shed some light on some non-economic factors that may also essential to FOSS collaboration.

¹⁰⁹ Lastowka, “Trademark Function”, *supra* note 73 at 1217

¹¹⁰ Lastowka comments: “If we see authorship simply as a system for efficiently parcelling out proprietary ownership rights, the law should grant ownership (denoting it as ‘authorship’) to the most efficient distributors and exploiters of works. Again, the problem with this model—from the standpoint of attribution—is that the non-statutory, non-dominant author lacks the control to secure attribution”. *ibid.*


services are not dissimilar from an authorial attribution system ascertaining the origin of creative works¹¹¹, though I will show soon the two systems are not exactly identical.

When a FOSS project has been able to provide a software product or service, whose quality can be consistently experienced by the public, then the name of this project may well accumulate enough reputation or goodwill to become a brand over the years.¹¹² Examples of brand-name projects are profuse and many of them already have had registered trademarks such as Linux®, Apache®, or DecoderPro®, just to name a limited few. The official guide provided by Software Freedom Legal Centre (SFLC) fully recognises the necessity of protecting a “brand”-name FOSS project through the trademark regime: “Like other products, FOSS applications develop reputations over time as users come to associate an application’s name with a particular standard of quality or set of features. Trade mark law can help protect this relationship of trust and reliance that a project develops with its users; it allows the project to maintain a certain amount of control over the use of its brand”.¹¹³ The Apache Software Foundation’s (ASF) is an outstanding example which has been serious about protecting the “Apache” brand by working out a trademark policy making clear what kind of marks and graphic signs that are intended to be protected: “‘Apache’, ‘Apache Software Foundation’, the multicoloured feather, and the various Apache project names and logos are trademarks of The Apache Software Foundation, and are usable by others only with express permission from the ASF.”¹¹⁴ Moreover, the ASF is also a pioneer that explicitly incorporates a trademark clause

¹¹¹ Lastowka proposes to do a thought experiment to see the connection: “If one were [...] to equate authorial attributions with trademarks and works of authorship with all other goods, misattributions would capture a situation that seems generally analogous to trademark infringement.” Furthermore, if misattribution is analogous to trademark infringement, plagiarism can be said to be “reverse passing off”. Greg Lastowka, “Trademark Function” supra note 73, at 1193

¹¹² Similarly, Rose observes that an author’s name can be a brand name: “the name of the author—or artist, conductor, director, or , sometimes, start, for in mass culture the authorial function is often filled by the star—becomes a kind of brand name, a recognizable sign that the cultural commodity will be of a certain kind and quality.” See Mark Rose, *Author and Owners—The Invention of Copyright*, (Cambridge, Mass. & London: Harvard University Press, 1993) pp.1-2

¹¹³ See Software Freedom Law Center, “Chapter 5: Common Trademark Issues” in *A Legal Issues Primer for Open Source and Free Software Projects*, 3 March 2008, p.31

¹¹⁴ Apache’s graphic mark of multi-colour feather looks like this:  **Apache** see Apache Software Foundation, “FAQ—Is Apache a Trademark”, at <<http://www.apache.org/foundation/licence-FAQ.html#Marks>>

into its licensing scheme, which is designed to prevent unauthorised use of the marks that it owns:

This [Apache] License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.¹¹⁵

The Apache-style trademark clause is widely used in the FOSS community. Artistic License 1.0 (as the one used by JMRI in the *Jacobsen* case) is another prominent example: “The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.”¹¹⁶ Interestingly, early versions of the GPL do not have a trademark clause and Stallman was not aware that it could have been an issue. It was not until the latest version 3.0 when Stallman decided to follow ASF’s footsteps by adding an option to decline “to grant rights under trademark law for use of some trade names, trademarks, or service marks”.¹¹⁷

FOSS trademarks can be either registered marks or unregistered marks, the former of which gives stronger and more certain protection. SFLC strongly recommends that FOSS projects to register marks with trademark authorities: “Registration grants much stronger protections for your trademark if someone else uses the mark in connection with goods similar to the ones described in your registration application.”¹¹⁸ For example, “Linux®” is a registered mark owned by Linus Torvalds¹¹⁹ and administered by the Linux Mark Institute (LMI)¹²⁰. It is worth noting

¹¹⁵ Section 6, Apache License 2.0

¹¹⁶ Section 8, AL 1.0

¹¹⁷ Section 7 (e), GPL 3.0

¹¹⁸ SFLC (Richard Fontana et. al.), *A Legal Issues Primer for Open Source and Free Software Projects*, 3 March 2008, at <<http://www.softwarefreedom.org/resources/2008/foss-primer.html>>

¹¹⁹ A search of “Linux” from the online USPTO Trademark Application and Registration Retrieval system (TARR) shows following information:

Mark (words only): LINUX

Standard Character claim: No

Current Status: This registration has been renewed.

Date of Status: 2005-11-29

Filing Date: 1994-08-15

Transformed into a National Application: No

Registration Date: 1995-09-05

that Torvalds and his fellow programmers were initially not aware of the need to register the “Linux” mark when the project started to take off in the early 1990s and the GPL 2.0 adopted by Linux has not yet had an explicit trademark clause. However, in 1994, William R. Della Croce, Jr., a person unrelated to the development of the Linux kernel, first registered the “Linux” mark in an attempt to collect licensing fee from various Linux distributors. In 1996, Linus Torvalds on behalf the Linux community filed a lawsuit against Croce’s bad-faith registration. The case was settled and it led Croce to transfer the ownership of the mark back to Torvalds, who then delegated his right to the LMI for the use of the mark. A similar dispute over a registered FOSS mark also took place in the aforementioned *Jacobsen* case. The JMRI project has registered mark “DecoderPro®”, but the domain name decoderpro.com was first registered by Matthew Katzer, who had never involved in the development of the DecoderPro product. According to Katzer, the reason for his registration was as follows: “If I decide that to released (sic) a licensed version of an open source development effort, what better place to have it [than] the name of the development effort?”¹²¹ Katzer’s reason, though hardly justifying his cyber-squatting behaviour, usefully indicates that the name of a FOSS project can be a valuable asset as it points towards to the “development effort” behind the project. Independent of his copyright claim, Jacobsen filed a Uniform Domain Name Dispute Resolution Policy (UDRP) complaint with the World Intellectual Property Organisation (WIPO) in Switzerland in order to regain the domain name for the DecoderPro project. The WIPO panel ruled that Katzer’s registration was in bad faith and “there is essentially a purpose on the part of Katzer to disrupt the business of a competitor by interfering with [JMRI team’s] exercise of [their] trademark rights”. The panel ruling led DecoderPro.com to be transferred.¹²² In summary, the above two disputes give an glimpse into the world where the registered marks (or domain names) may play a function of crediting FOSS projects because they can be extremely useful to ascertain their origin as well the development efforts behind.

(last retrieved 26 April 2009; the result is followed by a more detailed registration history which is not included here)

¹²⁰ See Linux Mark Institute at <<http://www.linuxmark.org/>>

¹²¹ See *JMRI Defense: Regaining DecoderPro.com* at <<http://jmri.org/k/UDRP/index.shtml>>

¹²² WIPO Arbitration and Mediation Center (Administrative Panel Decision), “WIPO finding on Robert G. Jacobsen v. Jerry R. Britton”, Case No. D2007-0763, July, 2007 at <<http://www.wipo.int/amc/en/domains/decisions/html/2007/d2007-0763.html>>

Apart from registered marks, it is also possible for the trademark regime to protect unregistered marks through the common law action of “passing off”. In the US context, it was not uncommon for US authors to invoke Section 43(a) of the Lanham Act, which has codified the “passing off” law, to protect authorial attribution to creative works¹²³ and even the artistic integrity of the authors.¹²⁴ In 2001, the US Court of Appeals for the Eleventh Circuit, in *Planetary Motion v. Techsplosion*, ruled in favour of an attempt to resort to Section 43(a) for the unregistered trademark protection of a FOSS product. In this case, Byron Darrah has written an email service program for UNIX users and he released it under GNU GPL 2.0 free of charge since 1994. He named this software “Coolmail”, which appeared with the announcement sent to the users and the user manual for each release. Darrah later transfers all intellectual property rights in this software to a company known as Planetary Motion, who then became the proprietor of “Coolmail” software. Techsplosion was another company, which in 1998 offered a similar email service program also bearing the mark “Coolmail” (four years after Darrah’s use of “Coolmail”). Planetary Motion sued Techsplosion for infringement of the unregistered mark that was purchased from Darrah under the Section 43 (a). The assignment of Darrah’s rights to Planetary Motion was not disputed, but Techsplosion contended that Darrah’s “Coolmail” software was merely a hobby unworthy of common law trademark protection in the first place.¹²⁵ The Eleventh Circuit found that Darrah did not “warehouse” or squat on the mark, but his continuous distribution of the software under this mark gave him the prior right to the “Coolmail” mark and his effort deserved to be identified as the source of the product. In particular, the distribution of the software under GPL 2.0, which lacks an explicit trademark clause, did not undermine Darrah’s IP rights in his software including the trademark. The court pointed out:

That the Software had been distributed pursuant to a GNU General Public License does not defeat trademark ownership, nor does this in any way compel a finding that Darrah abandoned his rights in trademark. Appellants [i.e.,

¹²³ e.g., *Follett v. New American Library* 497 F. Supp. 304 (SDNY, 1980) ; For a detailed critique of using Section 43 (a) to prevent misattribution or “reverse passing off”, see Roberta Rosenthal Kwall, “The Attribution Right in the United States: Caught in the Crossfire Between Copyright and Section 43(A)” (2002) *Washington Law Review* 985

¹²⁴ e.g., *Gilliam v ABC*, 538 F.2d 14 (1976)

¹²⁵ 261 F.3d 1188 at 1198

Techsplosion] misconstrue the function of a GNU General Public License. Software distributed pursuant to such a license is not necessarily ceded to the public domain and the licensor purports to retain ownership rights, which may or may not include the rights to a mark.¹²⁶

In fact, the GPL is more than just showing Darrah's non-abandonment of proprietary rights to the public domain, but it is evidence that positively affirms the intention to control the "Coolmail" mark. "Because a GNU General Public License requires licensees who wish to copy, distribute, or modify the software to include a copyright notice, the license itself is evidence of Darrah's efforts to control the use of the 'Coolmail' mark in connection with the Software".¹²⁷ Most relevant to our discussion, the court is also aware that the "Coolmail" mark is extremely useful for users to identify Darrah as the source, as well as the lead "Author/Maintainer", of the software:

[...] the mark served to identify the source of the Software. The 'Coolmail' mark appeared in the subject field and in the text of the announcement accompanying each release of the Software, thereby distinguishing the Software from the other programs that might perform similar functions available on the Internet or sold in software compilations. The announcements also apparently indicated that Darrah was the 'Author/Maintainer of Coolmail' and included his e-mail address.¹²⁸

This source/author identification function of the mark will become ever more important for identifying the origin of a FOSS project, because it may be subject to unlimited numbers of downstream redistributions. For example, when SuSE—a famous distributor of the Linux system in Germany—attempts to incorporate Darrah's software into its own product, it can easily locate Darrah as the "author/maintainer" of the software in the US and request permission to use the "Coolmail" mark. This will in turn help SuSE's users or even competitors to easily trace the origin of the "Coolmail" software in SuSE's redistribution. The appellate court observes: "Any individual using the SuSE product, or competitor of SuSE, that

¹²⁶ *ibid.*

¹²⁷ *ibid.*, FN 16 at 1198

¹²⁸ *ibid.*, at 1197

wanted to know the source of the [Coolmail] program that performed the e-mail notification function, could do so by referring to the user manual accompanying the product.”¹²⁹

It is not difficult to find that in a small project like “Coolmail”, Darrah is the author as well as the owner of the program. This author-ownership makes him the undisputable project leader who dictates the direction in which the program will go.¹³⁰ In this sense, Coolmail is a typical case of Weber’s “small project” where leadership is “essentially the same as ownership”.¹³¹ However, as has already been mentioned in Section 5.2.3, the basis of leadership in a much bigger project like the Linux kernel is very different from a small project like Coolmail, but it shifts from lead programmers’ ever-dwindling ownership of the program to their ever-growing stewardship responsibility in shepherding the project. It is worth noting that project leaders’ stewardship by no means entirely extinguishes their own, let alone ordinary contributors’, “IP” rights. Instead, one of their most important stewardship responsibilities is to coordinate programmers’ authorial interests in order to make a legally coherent project that can be attributed as a collective project-level “author”. I will now explain project leaders’ author-stewardship as a way to summarise FOSS programmers’ legal persona that has been affected by both copyright and trademark laws.

5.3.3 Legal Persona of Author-Stewardship

The legal persona of the FOSS authors is no doubt a complex and puzzling phenomenon. It is an extremely grey intersectional area where it is not always clear whether copyright or trademark laws should be invoked to regulate the designation of the authorial origin of FOSS programs. For example, in *Jacobsen*, FOSS attribution is protected through the copyright route, whilst, in *Planetary Motion*,

¹²⁹ The court notes that in the user manual for SuSE Linux 4.3 contains attributional information like this:

“Copyright (c) 1994 Byron C. Darrah

Author: Byron C. Darrah <darrah@kaiwan.com> , Randall K. Sharpe <rsharpe@ncsa.uiuc.edu>
Version 1.3”

See *ibid.*, FN 15 at 1197

¹³⁰ Darrah’s author-owner-leadership even gives him the authority to assign the whole program to the company Planetary Motion later.

¹³¹ Weber, *Success*, p.166

trademark law is used to do the similar job. In fact, it is important to see that FOSS authorship has both *copyright* and *trademark* elements, which play slightly different roles in FOSS projects and are worth explaining separately. To begin with, copyright is a legal institution that traditionally regulates the activities such as reproduction, modification and distribution of works of authorship. An author, from a traditional copyright's point of view, is an individual who impresses his creative personality onto his work. Ginsburg's definition of "author" nicely captures the essence of conventional copyright law's understanding of authorship:

[...] an author is (or should be) a human creator who, notwithstanding the constraints of her task, succeeds in exercising minimal personal autonomy in her fashioning of the work. Because, and to the extent that, she moulds the work to her vision (be it even a myopic one), she is entitled *not only to recognition and payment, but to exert some artistic control over it.*¹³² (added emphasis)

From the above definition, we see that one of the most important aspects of copyright is to reward an individual author with certain exclusive control over his creation (be it "recognition and payment" or "some artistic control over it"). Under Anglo-American copyright law, this exclusive control is mainly interpreted as the protection of authors' economic interests rather than that of their attributional interests. (To put it succinctly, it puts "payment and recognition" before "artistic control".) So FOSS authors have to write *copyright* licences to collaterally protect their authorial attribution as if it is of great *economic* consequence. The most prominent example in the FOSS world is the aforementioned *Jacobsen* case, where the appellate court argues that attribution as the licensing condition is to fulfil the economic goal of FOSS programmers as copyright holders.¹³³ In short, following the *Jacobsen* rationale, it is essential for FOSS programmers to wear their legal persona as copyright owners, who are assumed to be the economic utility maximisers, in order to directly claim authorship under a FOSS licence.

However, it would be much harder to employ the same *Jacobsen* rationale to further satisfactorily explain lead programmers' stewardship to coordinate or organise peer-

¹³² Jane C. Ginsburg, "The Concept of Authorship in Comparative Copyright Law", supra note 4, at 1064

¹³³ *Jacobsen v. Katzer*, 535 F.3d 1373 (Fed. Cir. 2008); see also supra sub-section 5.3.1

produced contributions into a project as a whole. In fact, many lead programmers work hard to play down their own individual importance in the project and emphasise their non-economic motives to lead the project. For example, Linus Torvalds has been famous for his self-deprecating manner through belittling himself as “basically a very lazy person who likes to get credit for things other people actually do”.¹³⁴ Also he refrains from expressly admitting that his devotion to the project is motivated by mainly maximising his individual economic utility or reputational incentive, but he regards the intrinsic pleasure to solve programming problems in a community as the primary motive.¹³⁵ In Torvalds’s own parlance, this intrinsic pleasure in coding is the “Entertainment with the capital E”, which is “the kind that gives your life meaning” among FOSS programmers.¹³⁶ Of course, it is FOSS project leaders’ stewardship responsibility to channel his and other programmers’ “Entertainment” into a meaningful collaborative effort. In short, lead programmers, as the anchorage of a project, need to build their credibility and trustworthiness from their relatively selfless commitment. Weber observes that Torvalds and leader programmers of other large projects must share this good stewardship quality of being humble and at the same time capable of motivating other programmers:

While leaders of other large projects have different personality traits, they do tend to share an attitude that underemphasizes their own individual importance in the process. And they share, more importantly, a commitment to invest meaningful effort over time in justifying decisions, documenting the reasons for design choices and code changes in the language of technical rationality that is the currency for this community.¹³⁷

¹³⁴ For example, Torvalds is honest about his ever diluted individual contribution in the Linux kernel and he makes this famous statement in his usual self-deprecating manner: Raymond, *Cathedral*, supra note 36

¹³⁵ Weber observes: “While [Torvalds] is not shy and does not deny his status as leader, he does make a compelling case that he was not and is not motivated by fame and reputation. The documented history, particularly the archived email lists, support him on this point. He continues to emphasize the fun of programming and opportunities for self-expression and claims ‘the feeling of belonging to a group that does something interesting’ as his principal motivation.” Weber, *Success*, p.167

¹³⁶ Linus Torvalds, “What Make Hackers Tick? a.k.a. Linus’s Law” as the prologue to *The Hacker Ethic and the Spirit of the Information Age*, by Pekka Himanen, (NY: Random House, 2001) p. xvi

¹³⁷ Weber, *Success*, p.167

Of course, FOSS leaders have an important stewardship responsibility to make sure that the collective authorship of the whole project is always correctly attributed. There are two routes to achieve this goal. The first route is to unify copyright ownership of individual programmers' code into the hand of lead programmer on behalf of the project. This would give the project leader undisputable power to enforce FOSS licensing conditions including its attribution requirement. For this reason, Software Freedom Law Centre (SFLC) believes “[c]entralizing copyrights via direct copyright assignment provides some compelling advantages if developers are willing to do so”.¹³⁸

However, it is not always an easy task to persuade every programmer to assign their copyright to the project despite some perceived advantages of doing so. So project leaders may seek to protect the collective authorship of a project via the trademark route, which seems to be a more convenient legal form that FOSS leaders' stewardship can fit into. There are two distinct features of this trademark-protected stewardship. On the one hand, trademark protects collective authorship of a whole project but not directly individual authorship; on the other hand, trademark does not only protect authors, but it also protects the public from being confused about the authorial origin of the software. First, almost all *individual* authors would get copyright over their individual contribution, but most of them are unlikely to have their names protected as trademarks. Only the names of the whole projects such as “Linux”, “JMRI” or “Apache” would be the depository of *collective* reputation or goodwill that merits trademark protection. Here the collective authorship of a project deviates from Ginsburg's definition of “author” as an individual “human creator”, but it is largely an organisational fiction that integrates countless individual authorship under a collective persona bearing the project's name. This fictional collective persona is necessary, because it is much easier for the public to identify

¹³⁸ “In general, the most important reason to contribute copyrights to the project is to enable the project to enforce the license. Unifying ownership of the copyrights gives the project indisputable enforcement power that is both simple and clear. If copyright ownership is scattered throughout a developer community spanning many countries and years, enforcement efforts face additional barriers. With a diluted base of copyright holders, enforcement efforts are hindered by figuring out which pieces were copied, tracking down the developers who contributed those pieces, and then getting them involved in the enforcement action. Especially in cases where it is unclear how much or which code has been copied, the project needs to avoid quibbling about whose copyrights are at stake.” SFLC, Chapter 2, SFLC (Richard Fontana et. al.), *A Legal Issues Primer for Open Source and Free Software Projects*, supra note 118

one FOSS product with one collective “author” rather than countless individual authors. The collective FOSS authorship bears out Heymann’s “authonym” theory that separates “the fact of authorship” and “the statement of authorship”, the latter of which is rightly called “authonym” by her.¹³⁹ So in a FOSS project, each individual authors would be the Ginsburgian “human creator” (i.e., the fact of authorship) while the collective author bearing the project’s name would be the Heymannian “authonym” (i.e., the statement of authorship), which is mainly used to give a unified persona that can be easily recognised by the public. For this reason, Heymann argues that “authonymic attribution is not a matter of authorial justice, but rather a matter of organizational integrity. It preserves the organizational framework that authonyms create such that reader responses will be informed and minimizes the likelihood of confusion a consumer of creative commodities might otherwise experience.”¹⁴⁰

Secondly, to get a FOSS project’ name correctly attributed is not merely a matter of garnering credit for this project, but it also protect *users from the public* from being confused about the authorial origin of the program. It is in these users’ interest to make them always go to the software bearing the name that can correctly identify the authorial origin of the project. Just as Heymann observes, when a work is misattributed, it does not only cause a copyright harm to the author but also a trademark harm to the public, who may well be confused about the origin of the creative work.¹⁴¹ In this sense, trademark law can be employed not just to give credit to the collective authors but it also prevents public confusion, because it is exactly a legal institution that regulates the designation of the sources of products or services. For example, the public deserve to know that the “DecoderPro” product is produced by Jacobsen’s team rather than Katzer’s company, or the open source “Coolmail” is originated from Byron Darrah rather than Techsplosion, or the Linux kernel product is maintained by Linus Torvalds and his colleagues rather than Mr. Croce. Based on the correct attribution, members of the public will be protected from using a “wrong” FOSS product bearing the same or similar marks.

¹³⁹ Laura A. Heymann, “The Birth of the Authonym: Authorship, Pseudonymity, and Trademark Law” (2005) 80 *Notre Dame Law Review* 1377 at 1446

¹⁴⁰ *ibid.*

¹⁴¹ *ibid.*, at 1383

Moreover, I should also warn that the legal form of FOSS author-stewardship through trademark does not necessarily reproduce the whole hacker stewardship tradition. Nor do the FOSS licences that prescribe the *minimum* stewardship responsibility to secure software freedom (which has been discussed in Chapter 3). In fact, it is very difficult to translate a full sense of stewardship obligation, which “blends an awareness of both externally endowed inspiration and the cyclical dimension of creative enterprise” summarised by Kwall¹⁴², into one single legal form (be it trademark or copyright). A high sense of stewardship has more to do with what the sociologist Richard Sennett calls the “craftsmanship” instinct, which is hard-wired to FOSS programmers’ fundamental motivational make-up. According to Sennett, the “desire to do a job well for its own sake” is a common hallmark of to all types of craftsmanship¹⁴³ and FOSS programmers are an exemplary type of these craftspeople¹⁴⁴. Just as a computer hacker in Levy’s book says: “Hackers can do almost anything and be a hacker. It’s not necessarily high tech. I think *it has to do with craftsmanship and caring about what you’re doing.*”¹⁴⁵ (added emphasis) It is exactly this high sense of stewardship/craftsmanship to do a job well for its own sake (or simply the feeling of “caring about what you’re doing”) that motivates many FOSS programmers, especially those long-term project leaders, to work for a certain project for a sustaining period of time. In this light, it is understandable for a few FOSS developers to call for replacing ownership with stewardship in understanding FOSS: “we must make a distinction between ‘ownership’ and ‘stewardship.’ Ownership is something that is fully transferable from one owner to another without loss of values. [...] Stewardship, on the other hand, applies when something undergoes change, when it evolves, or when it has some kind of life cycle.”¹⁴⁶ In this sense, stewardship is a better way of realising the full value of an evolving object that has a life cycle (e.g. animals or software that needs to be “herded”) than private

¹⁴² Roberta Rosenthal Kwall, “The Author as Steward ‘For Limited Times’”, (2008) *Boston University Law Review* 685 at 703

¹⁴³ Sennett, *The Craftsman*, supra note 58, p.9

¹⁴⁴ Sennett, in his study of craftsmanship in the western civilisation, finds that Linux programmers are not dissimilar from other traditional craftsmen such as carpenters since the time when the Homeric hymn to Hephaestus (i.e. master god of craftsmen) was written. He observes that people “who participate in ‘open source’ computer software, particularly in the Linux operating system, are craftsmen who embody some of the elements first celebrated in the hymn to Hephaestus” and “Linux draws on craftsmen in an electronic bazaar.” Sennett, *ibid.*, pp.24-25

¹⁴⁵ Levy, *Hackers*, p.434

¹⁴⁶ Chris DiBona, Danese Cooper, and Mark Stone, “Introduction” to *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O’Reilly, 2006) p. xxxvii

ownership, which might be only good at dealing with discrete static non-evolving objects. To assume steward's responsibility is not an easy job but it requires a lot of skills and competence and "only a good steward can realize the full value of that which is stewarded".¹⁴⁷ Most importantly, a carefully stewarded project tends to be nurtured by a long-term collaborative relation among programmers, and it is mostly likely coordinated by a highly capable long-term project leader, who would then always be associated with this project. This long-term stewardship would sometimes outcompete many commercial proprietary software projects, which are not "stewarded" but commercially "managed" by company executives:

The proof is in the longevity of open source software projects and the stewards who tend them. Linus Torvalds is still at the head of the Linux kernel 'tribe' more than a decade after the first public release of Linux. Eric Allman has guided Sendmail for more than 20 years. Larry Wall is still the guiding vision behind Perl, gain after more than 20 years. In these and many more cases, a common core group stood behind the software for *far longer than most proprietary software enjoys the benefits of a common development team*. It is this—the dynamics of stewardship—far more than the 'legions of programmers' that accounts for the success of open source software.¹⁴⁸ (added emphasis)

Raymond makes a similar observation about Stallman's long-term stewardship that gives the GNU Emacs project a "unified architectural vision", and most interestingly, it makes Stallman stewardship-"author" of the project:

In fact, there have been open-source projects that maintained a coherent direction and an effective maintainer community over quite long periods of time without the kinds of incentive structures or institutional controls that conventional management finds essential. The development of the GNU Emacs editor is an extreme and instructive example; it has absorbed the efforts of hundreds of contributors over 15 years into a *unified architectural vision*, despite high turnover and the fact that only one person (*its author*) has been

¹⁴⁷ *ibid.*

¹⁴⁸ *ibid.*, p. xxxviii

continuously active during all that time. No closed-source editor has ever matched this longevity record.¹⁴⁹ (added emphasis)

To summarise, the success of a collaborative FOSS project does not only depend on a good number of *individual* programmers, but lead programmers' good stewardship—which makes individually contributed code into a collective one—is an equally important matter. This stewardship mainly finds its legal form in trademark, which is most helpful to protect the collective authorship of the project as whole. However, trademark law does not translate the whole stewardship obligation into a particular legal form, but FOSS project leaders need to make additional efforts to coordinate the long-term collaboration under their stewardship.

5.4 Conclusion

This chapter has examined FOSS programmers' authorial persona in both aesthetical and legal senses. Aesthetically I find that the Romantic author vision of author-genius does not tally well with programmers' collaborative attempt to "author" a FOSS project. Instead, I find that FOSS programmers' desire to be identified as the authorial origin of their creation happens at both individual and collective levels. It is very important to recognise the role of project leaders/stewards who are crucial to channel *individual* contributions into a *collective* work of authorship, which can be held responsible and deserve credit for this FOSS project as a whole. Legally, the Anglo-American system does not readily recognise programmers' non-economic authorial interests including their right to be attributed as the origin of their creation. So FOSS developers have to wear the legal persona of copyright/trademark owners to indirectly claim their authorship. The situation is further complicated by the need to claim FOSS authorship at both individual and collective levels. I find that copyright licences can largely satisfy the need to recognise individual authorship, whilst trademark gives a more suitable legal form to reflect lead programmers' stewardship responsibility to defend FOSS projects' collective reputation and goodwill.

¹⁴⁹ Raymond, *Cathedral*, supra note 36

Chapter 6 Conclusion

6.1 Contributions to the Scholarly Literature

The emergence of free and open source software (FOSS) has posed many challenges to mainstream ways of producing and circulating software as proprietary products. This dissertation has been written in an attempt to make sense of only one dimension of these challenges: i.e. FOSS programmers' use of intellectual property licensing schemes in support of large-scale decentralised collaboration. On the surface, these FOSS licences may look quite similar to other mass-market standard-form contracts including proprietary software licences, where software users are given the licensing terms on a take-it-or-leave-it basis. However, my scrutiny of these licences in this dissertation shows that FOSS licences are different from their proprietary cousins in three aspects, the identification of which is intended to be my three modest contributions to the legal scholarship of software licensing jurisprudence. These three distinctions respectively cover the historical (Chapter 2), legal (Chapters 3 and 4) and authorial (Chapter 5) aspects of FOSS licensing. My study of these three aspects aims to create a synergy to show FOSS programmers' struggles against a dominant assumption—which has underpinned both intellectual property and contract laws—that human beings are fundamentally self-constituting individuals and they work mostly in a possessively individualistic and competitive environment. I will now briefly review each of the aspects that are essential to a sound understanding of the collaborative ethos in relation to FOSS licensing.

Firstly, FOSS licensing does not come into existence in a historical vacuum, but it is a unique product from a historical period when the MIT-style hacker custom was eclipsed by the rise of intellectual property regulation (especially copyright) over software in the early 1980s. The GNU General Public License (GPL) is often believed to be the very first conscious attempt to graft the hacker custom on the IP institution through a licensing scheme crafted by Richard Stallman. However, it is relatively a difficult task to gauge the exact influence of the lingering influence of the hacker custom in the GPL, which has also been criticised for overly relying on software copyright and its underlying proprietary ideology. In order to avoid

exaggerating the emergence of FOSS licensing either as a historical inevitability or merely the consequence of a few isolated one-off accidents, my assessment of the historical context is based on a few general prescriptive tenets of the Hacker Ethic¹ but it is also balanced out by some historically specific events including the Emacs dispute, which directly led to the creation of the first copyleft licence in 1985². I highlight three important controversies (i.e. the Xerox printer incident, the Symbolics incident and the Emacs incident) to show that FOSS licensing comes out of a mix of idealism and pragmatism. All of them share a common theme in that Stallman, as a dedicated hacker, has campaigned hard to rescue the Hacker Ethic rooted in its original MIT-based setting, where programmers are guided by their “Hands-on Imperative” to indulge their curiosity about computer technology.³ In 1998, this free software movement led by Stallman was further complicated by a spin-off campaign under the banner of “open-source” led by Eric Raymond to integrate non-proprietary software into the commercial mainstream. I argue that the “open source” twist both benefits and challenges Stallman’s cause. As a benefit, the “open source” agenda functions much like a business plan for “free software” to be marketed to a much wider constituency beyond the close-knit MIT-style hackerdom. As a challenge, it also forces Stallman to clarify his “free software” philosophy to put an ethical limit on the commercialism of “open source” movement by emphasising the intrinsic value of “software freedom”.⁴ My analysis also shows that the two definitional baseline documents—Free Software Definition (FSD) and Open Source Definition (OSD)—respectively championed by Stallman and Raymond are compatible with each other as both stipulate similar minimum stewardship responsibility for FOSS programmers to preserve software commons. In contrast, proprietary software developers mainly use their licence to maximise their revenue streams and there is no

¹ Levy has provided a definitive account of the Hacker Ethic containing six tenets. Steven Levy, *Hackers—Heroes of the Computer Revolution* (London: Penguin Books, 1984,1994)

² The Emacs dispute has been carefully detailed by Christopher Kelty, whose account of the story calls into question the real influence of so-called Hacker Ethic as identified by Levy. Kelty does not believe that the birth of the GPL is a purely ideologically driven product from the Hacker Ethic, but it largely a knee-jerk response to the specific dispute between Stallman and Gosling over the Emacs program from 1983 to 1985. In this sense, the Hacker Ethic, according to Kelty, may be an exaggerated influence, because this “vaunted” ethic only reveals itself in its “native practical setting, rather than as a rarefied list of rules”. See Kelty, *Two Bits--The Cultural Significance of Free Software*, (Durham: Duke University Press, 2008), p.15

³ This “Hands-on Imperative” is Tenet 1 of the Hacker Ethic documented by Levy.

⁴ Stallman, “Why Open Source Misses the Point of Free Software” at <<http://www.gnu.org/philosophy/open-source-misses-the-point.html>>

ethical limit to rein in that motive. They do not have a responsibility to make software reproducible, modifiable and redistributable for downstream users or developers. In other words, proprietary licences create an “asymmetrical relation” where there is an unbridgeable gap between software owners and the non-owning public. On the other hand, FOSS licences create a “symmetrical relation” where upstream and downstream developers have exactly the same sets of rights (i.e. software freedom) and obligations (i.e. the minimum stewardship responsibility as specified in FSD and OSD) to co-develop software.⁵

Secondly, the symmetrical relations intended by FOSS licensing do not mean anarchy but are organised around two legal institutions covering both “IP” and contract. As these two institutions provide different mechanisms to structure a licence, I need to deal with them separately, which will eventually lead to my proposal to tackle the issue from a relational contract perspective. First, FOSS programmers are not simplistically for or against “IP”, but they have much more nuanced understanding of the issue. They are aware that “IP” is not a unified body of law but that software freedom is affected by its two important components—copyright and patent—in subtly different ways.

Copyright: When copyright was first extended to software as though it were a kind of literary work, FOSS programmers’ initial knee-jerk reaction was very negative.⁶ However, they soon discovered that copyright’s threat to software freedom could be contained by appropriately crafted licensing terms. In particular, after his dispute with Gosling over Emacs, Stallman wrote the “copyleft” condition into his copyright licence, which allowed publicly released modifications and improvements of the original code to be shared with the community in order to mimic the hackers’ old share-alike tradition. However, FOSS programmers’ use of copyright does not mean that they embrace copyright without reservation, because most of them are still against stretching copyright to further cover non-literal (i.e., functional) elements of software. This is exemplified by the campaign led by

⁵ The distinction between “symmetrical relation” (in commons) and “asymmetrical relation” (in private property) is detailed by Benkler, *Wealth of Networks: How Social Production Transforms Markets and Freedom*, (New Haven: Yale University Press, 2006) p.143

⁶ For example, Stallman thought it was “blasphemous” to the Hacker Ethic by copyrighting software programs in the early 1980s. See Weber, *Hackers*, p.419

Stallman and his MIT colleagues to protest against Lotus's lawsuit to bring its non-literal user interface under copyright law.

Patent: Unlike software copyright, FOSS programmers were not immediately aware, let alone able to make an assessment, of the threat from patent to software freedom. In 1981 when the US Supreme Court allowed a software-related invention to be patentable in *Diamond v. Diehr*, the issue simply passed unnoticed by most FOSS programmers.⁷ However, the hidden threat from patent only gradually revealed itself in the early 1990s (almost ten years after *Diehr*).⁸ FOSS programmers find patent threat much more difficult to handle and it can be only partially contained by licensing schemes. This is because the patent system is much less intuitive than copyright and at the same time it is prohibitively expensive for most individual programmers who pursue FOSS merely as a hobby to get patents. Despite this difficulty, the latest revision of the GPL (v3.0) does make some efforts to deal with various patent issues but this would not change Stallman and his followers' patent abolitionist position.⁹

Apart from "IP" law, the second legal institution that heavily affects FOSS licensing is contract. However, it is not always clear whether a FOSS licence, or more specifically some of its conditions, have a contractual status. The heated debate about whether a FOSS licence is a pure property licence or a contractual licence is emblematic of this puzzling issue.¹⁰ One of the most vocal oppositions against

⁷ 450 U.S. 175 (1981)

The European equivalent of *Diehr* is EPO's ruling on *Vicom* in 1987 and it similarly had little publicity when the decision was made. See *Vicom/Computer-related invention*, T208/84 [1987] *EPOR* 74

⁸ For example, in September 1991, Stallman himself was forced to abandon a data compression program contributed by a volunteer programmer. This is because just about one week before a release of GNU software, Stallman accidentally found a newly issued patent that might "read on" this contributed compression program. See Stallman, "Patent Reform Is Not Enough" at <<http://www.gnu.org/philosophy/patent-reform-is-not-enough.html>>

⁹ For Stallman, software invention patents are landmines which are impossible to avoid unless they are stopped being produced. See Stallman, "The Dangers of Software Patents", 24 May 2004, a talk delivered at the University of Dublin, Trinity College, organised by Irish Free Software Organisation, transcript by Glenn Strong, Malcolm Tyrrell, Aidan Delaney and Ciaran O'Riordan at <<http://www.ifso.ie/documents/rms-2004-05-24.html>>

¹⁰ The difficulty to draw the line between property and contract is not unique to FOSS licensing, but it has been a persistent problem to a wider range of legal phenomena. As Merrill and Smith observe that property (in general) and contract are both "bedrock institutions of the legal system" but "it is often difficult to say where the one starts and the other leaves off." Thomas W. Merrill and Henry E. Smith, "The Property/Contract Interface", (2001) 101 (4) *Columbia Law Review* 773 at 774

treating FOSS licence as contract comes from the Free Software Foundation, which insists that the GPL is a pure property licence and not a contract. The reason behind this opposition stems from the need to distance the GPL from the kind of software licensing jurisprudence used by Easterbrook in *ProCD v. Zeidenberg*, where a standard-form licence was ruled *contractually* binding on an end user.¹¹ The *ProCD* decision and its model law progeny (i.e. Uniform Computer Information Transaction Act or UCITA) is based on a law-and-economics presumption that all that a licensee needs is to maximise his material wealth through non-negotiated standard-form licensing contract, which is achieved by reducing transaction cost. I argue that this *ProCD* jurisprudence has largely inhibited the further development of FOSS licensing jurisprudence from being in keeping with a few new theoretical breakthroughs in contract scholarship. Most significantly, I propose that FOSS licensing jurisprudence, if scrutinised in a contractual framework, needs to incorporate Macneil's relational contract theory (RCT), which is conspicuously absent in the legal literature about FOSS licensing. For the sake of completeness, I list all the three contractual approaches to show how RCT stands out from the classical and neoclassical approaches to software licensing.

Contract as consent (classical approach): The classical contract model bases its legitimacy on the idea that consent is obtained through a fully bargained process between parties. It assumes that there is a single moment when the minds of negotiators unequivocally meet and the total contractual obligation is thus “presentiated” into a present paper document that is fully binding on parties after that moment. Neither proprietary software licensing nor FOSS licensing fits neatly into this classical model, whose rigidity may limit this approach to be only heuristically useful in explaining contractual exchanges in an idealised textbook setting.

Contract as discrete product (neoclassical approach): The neoclassical model deviates from the classical model by marginalising the role of fully verbalised “consent” in contract formation. Instead, it takes a self-claimed “realist” position to recalibrate contractual exchanges against the neoclassical rationale of material

¹¹ 86 F.3d 1447 (7th Cir.1996)

wealth maximisation for individuals. The *ProCD* decision epitomises this neoclassical approach by pretending that end users' silence is an acceptance of the standard-form licensing terms, which are actually justified on the basis that they provide lowest price for consumers with reduced transaction cost. By doing so, the *ProCD* jurisprudence effectively creates a kind of discrete "contract-as-product"¹² or "legal-ware"¹³ as if the licensing terms are an integral "physical" feature of the licensed product. This dissertation argues that it is exactly this neoclassical variant (rather than the classical law) that has posed the greatest conceptual obstacle to understanding FOSS licensing contractually. The neoclassical view may reflect well what proprietary software licensors intend to achieve, but it hardly explains the highly collaborative relations that are essential to the success of FOSS projects.

Contract as relation (RCT approach): Both classical and neoclassical approaches conceive of contractual exchanges as discrete transactions, where a sustaining relation developed between exchangers is not essential. (The only difference between the two is that the neoclassicist is willing to sacrifice the classicist consent for thorough transactional discreteness.) In contrast to contract-as-consent classicism and contract-as-product neoclassicism, I suggest that FOSS licensing should be understood as a relational contract where the software code is not traded merely as commodity but a kind of "relation-ware" to sustain long-term collaboration where participants are motivated by a multiplicity of values. The rationale behind this "relation-ware" is different from the current dominant *ProCD* law in software licensing jurisprudence in two senses. First, FOSS "relation-ware" still respects contract as a consensual relation, but parties' consent is now relationally understood in a longer-term context, where no total obligation is formed at one particular single moment. Instead, parties' consent is allowed to evolve when the project move on to reflect the highly serendipitous and flexible nature of FOSS contributions. In other words, no total obligation can be presentiated in the beginning of a FOSS project, but only a minimum set of responsibilities of programmers is written down in the text of a FOSS licence and

¹² Margaret Jane Radin, "Humans, Computers, and Binding Commitment" (1999) 75 *Indiana Law Journal* 1125 at 1126

¹³ See Michael J. Madison, "Legal-ware: Contract and Copyright in the Digital Age" (1998) 67 (3) *Fordham Law Review* 1025

they are mainly about how to keep software freedom rather than the actual content of contributions. Secondly, to make FOSS “relation-ware” is driven by a multiplicity of motivational forces ranging from the hard-core monetary motive to the more ambivalent reputational incentive to the “soft” values such as “software freedom” to the intrinsic satisfaction from coding, while the *ProCD* ruling tends to reduce this multiplicity to a single individualistic utility-maximising rationale. This second argument reflects what relational contract scholars are keen to achieve¹⁴, and is deserving of more attention from the academia, and FOSS licensing provides an excellent opportunity to prosecute such an endeavor. Based on the above two insights, I then propose to examine the GPL as an “umbrella relational contract”¹⁵, which coordinates many contributors’ legal commitments to a project. The GPL as an umbrella “relation-ware” is a compromise between two needs. On the one hand it tries to satisfy the need for *serendipity and flexibility* in terms of the actual content of contribution, which is not presentiated at all in the beginning. On the other hand, it also tries to cater to the need for limited *certainty* to make sure all generations of contributions would be legally compatible with each other in any downstream distribution. Furthermore, to analyse GPL as a relational umbrella contract also gives a chance to see how far RCT can be applied to a real-world collaborative situation. It hopes to show that RCT is not merely a scholarly thought experiment, but it may also provide judiciaries with some insights into some highly relational cases, where classical and neoclassical law designed for discrete transactions clearly cannot cope well.

The third contribution that I try to make is about FOSS programmers’ authorial consciousness as manifested respectively in their aesthetical and legal personas. In terms of FOSS authors’ aesthetical persona, there is little doubt that the Romantic aesthetical vision of author as solitary “genius” does not suit the highly collaborative nature of FOSS programming. It is inadequate in the sense that it shares the exactly the same individualistic presumption adopted by the discrete transactional view from the classical contract model as discussed above. In fact, in a FOSS project, there are

¹⁴ William C. Whitford, “Ian Macneil’s Contribution to Contracts Scholarship”, (1985) *Wisconsin Law Review* 545

¹⁵ For the phenomenon of “umbrella relational contract” in a general context, see Stefanos Mouzas, and Michael Furmston, “From Contract to Umbrella Agreement” (2008) 67(1) *Cambridge Law Journal* 37

not merely individual programmers who can be identified as the *individual authors* of their contributions, but more importantly, there is also a *collective author* that can be held responsible and deserve credit for the production of an integrated FOSS project as a whole. I thus argue that a full evaluation of FOSS authorship in relation to FOSS licensing should be scrutinised at both individual and collective levels, though the existing literature does not tend to be discerning enough to differentiate the two. In particular, I analyse the pivotal role of lead programmers in “stewarding” FOSS projects for a sustaining long period of time. These project-leaders’ author-stewardship does not replace the individual programmers’ efforts in actually producing code, but it only channels individual authorship into collective authorship of a certain project.

My enquiry of FOSS programmers’ legal persona tackles the following question: How do FOSS programmers claim credit through law? In short, FOSS programmers need to wear the legal persona as the “IP” owners of their contributions in the first place and then indirectly claims their authorship in order to compensate for the lack of statutory attribution right under the Anglo-American system. This may be achieved either via copyright or trademark for different situations.

Copyright: Unlike the continental European legal system, Anglo-American copyright does not readily recognise a standalone paternity right for software programmers to be attributed to their works. As a makeshift solution, copyright licences need to be crafted to make the attributional interests ride on the proprietary rights owned by FOSS developers. Just as Lastowka observes that copyright protects attribution only “in a collateral fashion” by using the device of its licensing schemes.¹⁶ So FOSS developers, in order to have their attribution right enforceable under law, must take on the legal persona as the *copyright owners* to begin with. This observation has been corroborated in a 2008 landmark ruling in *Jacobsen v. Katzer*, where the US Court of Appeals for the Federal Circuit (CAFC) enforced the condition in a FOSS licence that requires correct attribution to the original FOSS contributors.¹⁷ Although *Jacobsen* is widely hailed

¹⁶ Greg Lastowka, “The Trademark Function of Authorship”, (2005) 85 *Boston University Law Review* 1172 at 1214

¹⁷ 535 F.3d 1373 (Fed. Cir. 2008)

as a long-awaited triumph within the FOSS community, it is far from unproblematic. It is worried that the *Jacobsen* ruling only strengthens copyright owners' proprietary interest that becomes an unavoidable prelude to enforcing programmers' attribution right collaterally through a copyright licence. Furthermore, the licensing condition made by copyright owners can effectively allow a privately legislated moral right regime that upsets the initial balance intended by the copyright legislation. I think that the only permanent solution to solve this problem is a legislative change that separates out programmers' paternity right from their economic right and as a result FOSS programmers would no longer pretend that attribution furthers the economic goal of copyright holders under a copyright licence but it could be enforced in its own right.

Trademark: Apart from relying on copyright for collateral protection of attribution, many FOSS developers also actively seek trademark protection of their projects' names. This is because trademarks designating the origin of goods or services are not dissimilar from an attribution system ascertaining the authorial origin of creative works. The name of a project is worth protecting when it accumulates enough reputation or goodwill to become a brand name. Projects do not have to register their names with trademark authorities, though registration would give them stronger and more certain protection. The Anglo-American system brings unregistered marks under protection through the common law action of "passing off". In the US, it is not unusual to invoke Section 43(a) of Lanham Act, which codifies the "passing off" action, as a proxy paternity right to get authorial attribution. The 2001 US case *Planetary Motion v. Techsplosion* is exactly a successful example where a FOSS project had its unregistered mark (i.e. its project name "Coolmail") protected under the Lanham Act.¹⁸ Furthermore, I argue that, in a large-scale FOSS project, lead programmers have the stewardship responsibility to defend the collective reputation or goodwill of a project as a whole, and trademark lends itself to be suitable legal form to manifest this stewardship. There are two features of trademark protection of the name a FOSS project under stewardship. Firstly, trademark protects the collective authorship of a whole project but not directly individual authorship. Secondly, when a FOSS project is

¹⁸ 261 F.3d 1188

misattributed, it does not only cause a copyright harm to its author but also a trademark harm to the public, who can be confused about the authorial origin. In this sense, trademark law can be employed not just for the purpose of allocating credit (and possibly reputational incentives) to authors of their works, but it may also help the public to find a FOSS product with the right origin.

In summary, both copyright and trademark may be employed to protect FOSS programmers' attributional interests. Neither of the two regimes is entirely satisfactory because they make attribution heavily dependent on the strong proprietary right afforded by law in the first place. In particular, trademark gives a legal form to FOSS project-leaders' stewardship, which is responsible to defend the collective reputation or goodwill of the project as a whole, though it does not translate the whole hacker tradition to coordinate the collaborative efforts among FOSS programmers.

6.2 Avenues for Future Research

This dissertation is a study of some key legal issues concerning FOSS licensing jurisprudence, which is largely informed by Steven Levy's pioneering work on the Hacker Ethic as published in 1984 (one year before the "copyleft" licence was first invented by Stallman). In following decades, this Hacker Ethic has undergone a chequered development largely due to the changing legal environment concerning intellectual property regulation over software innovation. However, two more recent developments, which may have a continuous impact on the Hacker Ethic as well the FOSS movement, should not go unnoticed. One is the increasing corporate participation in FOSS projects and the other is the spilling over of the Hacker Ethic into non-programming or mixed innovations. I will now explain briefly why these two developments can be two avenues leading to worthwhile research in the future.

The first avenue concerns a reevaluation of Tenet 3 of the Hacker Ethic—"Mistrust Authority—Promote Decentralisation"—which was interpreted by Levy as hackers' mistrust of any type of centralised bureaucratic system¹⁹ epitomised by software

¹⁹ "Bureaucrats hide behind arbitrary rules (as opposed to the logical algorithms by which machines and computer programs operate): they invoke those rules to consolidate power, and perceive the constructive impulse of hackers as a threat." Levy, *Hackers*, p.41

companies like the IBM.²⁰ Levy argues that IBM programmers are “priests and sub-priests” and they “could never understand the obvious superiority of decentralized system, with no one giving orders.”²¹ Ironically, over two decades later after this observation was first made, IBM today becomes one of the most active companies contributing to FOSS projects including Linux and Apache.²² Apart from IBM, other corporate giants such as Google, Intel, HP, Novell, Red Hat, which would no doubt be classed as “bureaucracies” by Levy’s 1984 standard, are also important FOSS contributors.²³ Lerner and Schankerman, in a recent book-length research, further demonstrate that many companies in fact produce both proprietary code and FOSS code, the two of which can be closely “comingled” in a corporate environment.²⁴

In this light, it is important for scholars to examine the extent to which this corporate foray into FOSS would challenge Benkler’s peer-production model where code is produced independent from a hierarchical corporate structure oriented towards making economic profits.²⁵ In other words, it is worth finding out the degree of compromise that those employed FOSS programmers can afford to make without losing their independent status to the corporate culture.²⁶ Chapter 3 of this dissertation has provided a glimpse of this issue through the lens of “open source

²⁰ “The epitome of the bureaucratic world was to be found at a very large company called International Business Machines—IBM.” *ibid.*

²¹ Levy, *Hackers*, p.42

²² IBM, “Open Source at IBM” at <<http://www-03.ibm.com/linux/ossstds/oss/ossindex.html>>

²³ A survey conducted by the Linux Foundation shows that Linux contributors are not simply individual hobbyists, but a significant number of them have corporate affiliations. Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update* at <<http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf>>

²⁴ Josh Lerner and Mark Schankerman, *The Comingled Code: Open Source and Economic Development* (Cambridge, Mass.: MIT Press, 2010)

Although Lerner and Schankerman’s research has been praised for its unprecedentedly wide scope (surveying about 2300 companies and nearly 2000 programmers), it only represents a starting point of the still poorly understood corporate FOSS phenomenon and “the literature of this important development in recent economic history is still far from complete.” See *The Economist*, “Untangling Code”, reviewing Lerner and Schankerman’s book, 15th January 2011 at 79

²⁵ Yochai Benkler, “Coase’s Penguin, or, Linux and ‘The Nature of the Firm’” (2002) 112, (3) *Yale Law Journal* 369

²⁶ Raymond observes that some “star” FOSS programmers are likely to attract corporate “patronage” to support themselves financially. For example, the Linux Foundation (comprising mostly corporate developers and distributors of the Linux kernel) is able to pay Linus Torvalds a full salary and health insurance and thus saves him from having another day job. See Eric Raymond, “Open R&D and the Reinvention of Patronage” in *The Magic Cauldron*, 1999 at <<http://www.catb.org/~esr/writings/magic-cauldron/>>

patents”²⁷, though it is not intended to be a full account of the still emerging corporate FOSS phenomenon. It observes that some corporate FOSS developers are keen to build patent portfolios to defend themselves against potential patent litigation, whilst individual FOSS hobbyists are unlikely to do so given the sheer cost of getting and maintaining patents. Again IBM is a most conspicuous example of a corporation both developing FOSS and owning a large number of “software patents.”²⁸ Another interesting example is a FOSS patent consortium known as Open Invention Network (OIN) formed by corporate FOSS developers to defend Linux from patent litigation.²⁹ To acquire defensive patents for FOSS projects is far from a satisfactory solution because it does not eradicate the threat to software freedom from its root, i.e. the legal system that produces software invention patents in the first place. It also unfortunately creates a schism within the FOSS community into two divisions: one belongs to the well-financed corporate developers who are less interested in changing the patent system and the other belongs to hobbyist-developers with no direct corporate affiliation who are keener to defend software freedom in its own right. I think that FOSS licensing schemes would play a very limited role in eliminating this schism by reining in corporate penetration into FOSS. My speculation is that how far this corporate foray into FOSS will go would largely be dependent on the scope of the commercial success that these companies can achieve by selling their FOSS products on the market. Corporate FOSS is likely to flourish mainly in the consumer-goods area where products are mainly used by non-sophisticated end-users who are not expected to make modification or improvement to the software. Corporate FOSS would thus understandably be less able to harness large-scale decentralised collaboration under the peer-production model.³⁰ In this light, I argue that legal scholarship needs to be more attentive to this new development of corporate

²⁷ Leveque and Ménière call this phenomenon “open source patents” to account for patents owned by corporate FOSS developers. See Francois Leveque and Yann Ménière, “Copyright Versus Patents: The Open Source Software Legal Battle” (2007) 4(1) *Review of Economic Research on Copyright Issues* 27 at 42

See also Section 3.4, Chapter 3 of this dissertation for a more detailed analysis.

²⁸ In 2005, the IBM signalled its commitment to the FOSS cause by its pledge not to assert its 500 patents to the FOSS community. See IBM, “IBM Statement of Non-Assertion of Name Patents against OSS” at <<http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf>>

²⁹ See OIN’s website at <<http://www.openinventionnetwork.com/>>

³⁰ In particular, established companies may well use FOSS as a marketing gimmick to get quick publicity for their new products in a short space of time, rather than attracting large-scale collaboration with hobbyist developers. Google’s open-source Chrome browser and its Android smart phone platform are both good examples of this kind of corporate FOSS strategy.

participation in FOSS, which would be significant in changing the landscape of FOSS collaboration.

Apart from calling for more research into corporate FOSS, I also need to highlight another avenue that merits further research. This second avenue concerns the last tenet (i.e., Tenet 6) of the Hacker Ethic—“Computers can change your life for the better”—under which Levy predicts that the Hacker Ethic would spill over into and eventually benefit the non-programming world enabled by computer technologies: “Surely everyone could benefit from a world based on the Hacker Ethic. This was the implicit belief of the hackers irreverently extended the conventional point of view of what computers could and should do—leading the world to a new way of looking and interacting with computers.”³¹ Following this tenet, there seems no significant conceptual barrier preventing FOSS programmers from bringing their software freedom to other creative spheres such as music.³² A most glaring success story of this kind of endeavour is the attempt to build a collaborative online encyclopedia universally accessible to and modifiable by every internet user. Stallman, in an essay titled “The Free Universal Encyclopaedia and Learning Resource”, calls for “a universal encyclopedia covering all areas of knowledge, and a complete library of instructional courses” and “a conscious effort to prevent deliberate sequestration of the encyclopaedic and educational information on the net.”³³ This vision indeed led to the creation of *Wikipedia* under the efforts of Jimmy Wales and his collaborators, who use the wiki technology to enable users all over the world to create a universally free encyclopedia.³⁴ The *Wikipedia* phenomenon, clearly a product of the last tenet of the Hacker Ethic, also reflects a widespread optimism about “collective creativity” enabled by networked computer technologies (e.g. wiki), which are sometimes romanticised as the “weapons of mass collaboration” in the “age of participation”.³⁵

³¹ Levy, *Hackers*, p.46

³² Moglen argues that “music, and movies, and train schedules, and all other useful forms of information in the twenty-first century” are no more than “other types of software”. Eben Moglen, “Freeing the Mind: Free Software and The Death of Proprietary Culture”, 29 June 2003, <<http://moglen.law.columbia.edu/publications/maine-speech.html>>

³³ Richard Stallman, “The Free Universal Encyclopaedia and Learning Resource”, at <<http://www.gnu.org/encyclopedia/free-encyclopedia.html>>

³⁴ See Larry Sanger, “The Early History of Nupedia and Wikipedia: A Memoir”, in *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O’Reilly, 2006)

³⁵ The terms are used by Tapscott and Williams who argue: “New low-cost collaborative infrastructures—from free Internet telephony to open source software to global outsourcing platforms—allow thousands upon thousands of individuals and small producers to cocreate products,

Most significantly, this optimism has also led some lawyers to experiment with new licensing schemes to facilitate the *Wikipedia*-type collective creativity. The most well-known example is no doubt a set of creative commons licences that aim to “build a layer of content, governed by a layer of reasonable copyright law, that others can build upon” in a free and re-mixable culture.³⁶ However, some other lawyers are not entirely convinced by this trend. For instance, Merges, who is keen to defend private property in the digital era, is vehemently against over-romanticising “collective creativity” enabled by networked computer technologies. For Merges, the “weapon of mass innovation” would defeat neither individual creativity³⁷ nor private property.³⁸ I think that Merges is mostly right in the sense that creativity in the real world does not have one single particular mode but it can run the whole gamut from being very solitary to highly collaborative, though most contributions to a FOSS-inspired collective work are most likely to be closer to the collaborative end of the spectrum.

However, in order to further test Merges’ thesis that private property is still relevant to collective creation in the digital age, I think that more study needs to be done because the property system, on which FOSS licensing schemes are based, is by no means the only parameter that makes collaboration take place.³⁹ For this reason, I wish to narrow this enquiry down to a case study of a collaborative project known as PureData—which is a widely used computer music language—to see how far the Hacker Ethic (as well as Merges’ thesis about property) can stand in an intersectional area of programming and non-programming (i.e. musical) creativities. There are two

access markets, and delight customers in ways that only large corporations could manage in the past. This is giving rise to new collaborative capabilities and business models that will empower the prepared firm and destroy those that fail to adjust.” Tapscott and Williams, *Wikinomics* (London: Portfolio, 2006) p.11

³⁶ Lessig explains: by using creative commons licences, “[v]oluntary choice of individuals and creators will make this content available. And that content will in turn enable us to rebuild a public domain.” Lessig, *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity* (New York: The Penguin Press, 2004) p.283

³⁷ Merges does not believe that “collective works will and should systematically replace individual works in the digital era.” Robert Merges, “The Concept of Property in the Digital Era” [2008] 45 (4) *Houston Law Review* 1239 at 1249

³⁸ Merges argues that “property rights still make sense as a legal and social institution. [...] continuing to grant and enforce property rights does not threaten the viability of collective creativity, but [...] seriously *curtailing* property rights so as to further promote collective creativity *would* significantly undermine the conditions for individual creativity.” (original emphasis) *ibid.*

³⁹ Again, it would be inappropriate to solely credit private property for the making of collective works while ignoring other parameters contributing to the real lived experience of collaboration.

reasons why PureData is a promising candidate for this kind of research in the future. First, PureData sits astride two creative fields covering both software and music. On the one hand, it is a visual programming language with a modular and extensible architecture, whose lead developer/coordinator—Miller Puckette—is deeply sympathetic with the original MIT-type hacker ethic since he was an undergraduate student at MIT. In fact, the proprietary predecessor to PureData called Max, which was also initially developed by Puckette in 1988 when he was affiliated with Institut de Recherche et Coordination Acoustique/Musique (IRCAM) in Paris, has drawn many ideas directly from researchers and developers based at MIT.⁴⁰ Puckette later felt deeply disaffected by IRCAM's decision to strengthen intellectual property control over Max as a proprietary product because this created a lot more difficulties for Puckette and his colleagues to disseminate Max-related works to the world outside IRCAM.⁴¹ As a result, Puckette left IRCAM and started the spin-off PureData project licensed under a FOSS licence.⁴² On the other hand, PureData is not merely for software programmers but it also used by musicians dedicated to making electronic arts. It follows a long line of pursuit to build a kind of “composition machine”, which can be stretched back to the early modern time of Leibnitz (1646-1716) and Marin Mersenne (1588-1646).⁴³ Note that PureData as a versatile programming tool does not only facilitate electronic music making, but it also deals with other forms of electronic arts such as video and still images with its Graphics Environment for Multimedia (GEM) external, which also effectively bears out Tenet 4 of the Hacker Ethic: “You can create art and beauty on a computer”. In short, PureData users/co-developers are two categories of creators—programmers and sound artists—rolled into one. The second reason why PureData is worth further researching is that its community members are well aware of the ongoing debate about IP and creativity and consciously pursue their electronic arts in the spirit of the FOSS movement. Within the PureData community, there has been a palpable anti-

⁴⁰ Miller Puckette, “Who Owns Our Software—A First-person Case Study”, *2004 ISEA Online Proceedings*, available at <<http://crca.ucsd.edu/~msp/Publications/isea-reprint.pdf>>

⁴¹ Puckette's situation at IRCAM at this point is not dissimilar from Stallman's at MIT AL Lab in the early 1980s.

⁴² This licence is called Standard Improved BSD License (SIBSD), which is a variant of the original BSD License.

⁴³ For the intellectual and historical background of PureData, See Winfried Ritsch, “Does Pure Data Dream of Electric Violins?—PD Introduction and Overview” (Wolke Verlag, Hofheim, 2006) from the edited book based on the First International Pd-Convention 2004, Graz/Austria, p. 11

property sentiment akin to early computer hackers' dislike of private ownership of software⁴⁴. For example, Puckette himself argues that electronic arts (such as the PureData project) are not privately "ownable" when detached from physical embodiments:

Artifacts of art may be owned, but 'digital art' itself is not intrinsically *ownable* by anybody. This is bad news to composers, for instance, who obviously would like to own their scores. They do indeed own the paper and ink on top of it, but *the work exists only as a way of arranging things, not in the things themselves, and therefore can't be owned*. Composers and other digital artists must survive *by the mechanism of attribution*. This is indeed how J.S. Bach operated; the intervening years, dominated by physical printing presses and their output, can be seen as an aberration, now coming to an end.⁴⁵ (added emphasis)

I think that Puckette's above argument indicates at least two directions in which the so-called "digital art" may go under the impact of the MIT-style Hacker Ethic. Firstly, it can be read as a challenge to Merges' defence of private ownership in the digital age. Largely due to the nature of digital art "as a way of arranging things, not in the things themselves", this art form cannot be owned like physical objects. Puckette further suggests that the material gain from private ownership would distort the creators' (including both researchers and artists) self-motivation to indulge their academic or artistic passion: "It is now ironic that researchers and artists now find themselves trapped by their own efforts to make their creations have monetary value in the form of IP. Researchers [...] are too easily seduced by the promise of material gains to be reaped from our work. Artists [...] fall into the same trap. Both eventually lose control over their own work."⁴⁶ This argument bears strong resemblance to Stallman's polemic where he argues why software should not be owned.⁴⁷ Secondly, Puckette envisions a mechanism that protects digital artists' attributional interest to sustain their creation in the digital era just like in J.S. Bach's time. This argument is largely in line with my observation made in Chapter 5, where I find FOSS

⁴⁴ This dislike is strongly registered by Stallman in his essay "Why Software Should Not Have Owners", 1994, at <<http://www.gnu.org/philosophy/why-free.html>>

⁴⁵ Miller Puckette, "Who Owns Our Software—A First-person Case Study", supra note 40

⁴⁶ *ibid.*

⁴⁷ Richard, Stallman, "Why Software Should Not Have Owners", 1994, at <<http://www.gnu.org/philosophy/why-free.html>>

programmers are keen to claim the attribution right independent from the economic right of their works. It also converges with a growing legal literature calling for separating non-economic authorial right from private ownership right in creative works. For example, the legal scholar Zimmerman argues that it is possible to protect and encourage “authorship without ownership” in the current digital age.⁴⁸ I believe that more empirical research is necessary to examine the extent to which PureData artists’ practice can flesh out Zimmerman’s thesis (as opposed to Merges’s thesis) in a mixed creative environment of programming and arts.

6.3 Concluding Remarks

This chapter summarises three modest contributions that I intend to make to the existing literature of FOSS licensing jurisprudence. Firstly, it shows the historical context from which FOSS licensing emerged as a response to the rise of intellectual property regulation over software innovation. Secondly, it deals with the legal aspect of FOSS licences, which are proposed to be scrutinised under a relational contract perspective. The third contribution concerns the authorial aspect of FOSS licensing, which is shown to have developed independently from Romantic aesthetics. It explains the FOSS authors’ attributional right—in the legal form of copyright and trademark—at both individual and collective levels. Furthermore, I suggest that there are two possible avenues for future research. Firstly, more research needs to be done to assess the impact of the increasing corporate penetration in FOSS, which may gradually erode Benkler’s peer-production model of FOSS production. Secondly, I call for further research into the impact of the Hacker Ethic and IP law on the intersectional areas of programming and non-programming creativities and I suggest that an electronic arts project known as PureData is a promising candidate for continuing the line of inquiry of the FOSS movement in a broader context.

⁴⁸ Based on her observation that the Victoria literary publishing in 19-century England depended much more on authors’ relation with editors and publishers than the private property system, Zimmermann proposes that internet publishing can similarly take off without private copyright ownership. Diane Leenheer Zimmerman, “Authorship without Ownership: Reconsideration Incentives in a Digital Age” (2003) 52 *DePaul Law Review* 1121

Bibliography

Andrews, Cathy Joanne, *Bridging the Divide—An Exploration of Ian Macneil's Relational Contract Theory and Its Significance for Contract Scholarship and the Lived World of Commercial Contract*, PhD Thesis, (London: Birkbeck College, University of London, 2010)

Bainbridge, David, *Legal Protection of Computer Software* (Heywards Heath, West Sussex: Tottel Publishing, 2008, 5th Ed.)

Barnett, Randy E., "Conflicting Visions: A Critique of Ian Macneil's Relational Theory of Contract", (1992) 78 (5) *Virginia Law Review* 1175

Barnett, Randy E., "Consenting to Form Contracts" (2002) 71 *Fordham Law Review* 627

Beatson, J., *Anson's Law of Contract* (Oxford: OUP, 2002, 28th Edition)

Benkler, Yochai "Coase's Penguin, or, Linux and 'The Nature of the Firm'" (2002) 112, (3) *Yale Law Journal* 369

Benkler, Yochai, *Wealth of Networks: How Social Production Transforms Markets and Freedom* (New Haven: Yale University Press, 2006)

Bently, Lionel and Sherman, Brad, *Intellectual Property Law* (Oxford: OUP, 2009, 3rd Ed.)

Bergquist, Magnus and Ljungberg, Jan, "The Power of Gifts: Organizing Social Relationships in Open Source Communities", (2001) 11 *Information Systems Journal* 305

Bern, Roger C., " 'Terms Later' Contracting: Bad Economics, Bad Morals, and a Bad Idea for a Uniform Law, Judge Easterbrook Notwithstanding", (2003-2004) 12 *Journal of Law and Policy* at 772

Biancuzzi, Federico, "A Look Back at 10 years of OSI", 12 February 2008, at <<http://www.onlamp.com/pub/a/onlamp/2008/02/12/a-look-back-at-10-years-of-osi.html>>

Boyle, James, *Shamans, Software, and Spleens—Law and the Construction of the Information Society* (Cambridge, Mass.: Harvard University Press, 1996)

Boyle, James, "The Second Enclosure Movement and the Construction of the Public Domain", (2003) 66 *Law and Contemporary Problems* 33

Boyle, James, "Cultural Environmentalism and Beyond" (2007) 70 *Law and Contemporary Problems* 5

Boyle, James, *The Public Domain—Enclosing the Commons of the Mind* (New Haven& London: Yale University Press, 2008)

Burk, Dan, "Copyrightable Function and Patentable Speech" (2001) 44 (2) *Communications of the ACM* 69

- Burk, Dan, “Anticircumvention Misuse” (2003) 50 *UCLA Law Review* 1095
- Campbell, David, and Harris, Donald, “Flexibility in Long-Term Contractual Relationships: The Role of Co-operation”, (1993) 20 (2) *Journal of Law and Society* 166
- Campbell, David, “Ian Macneil and the Relational Theory of Contract” in Ian Macneil, *The Relational Theory of Contract: Selected Works of Ian Macneil*, ed. by David Campbell, (London: Sweet & Maxwell, 2001)
- Campbell, David and Collins, Hugh, “Discovering the Implicit Dimensions of Contracts”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. by David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003)
- Canfield, Kenneth, “The Disclosure of Source Code in Software Patents: Should Software Patents be Open Source?” (2006) VII *The Columbia Science and Technology Law Review*
- Chen-Wishart, Mindy, *Contract Law* (Oxford: OUP, 2008, 2nd Edition)
- Clapes, Anthony L., Lynch, Patrick and Steinberg, Mark R., “Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs” (1987) 34 *UCLA Law Review* 1493
- Collins, Hugh, “Introduction: The Research Agenda of Implicit Dimensions of Contracts”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. By David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003)
- Cooke, Elizabeth, *The Modern Law of Estoppel* (Oxford: OUP, 2000)
- Covotta, Brian and Sergeeff, Pamela, “ProCD, Inc. v. Zeidenberg” (1998) 13 *Berkeley Technology Law Journal* 35
- Dempsey, Bert J., Weiss, Debra, Jones, Paul and Greenberg, Jane, “Who Is an Open Source Software Developer?—Profiling a Community of Linux Developers”, (2002) 45 (2) *Communications of the ACM* 67
- DiBona, Chris, Cooper, Danese, and Stone, Mark, “Introduction”, in *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O’Reilly, 2006)
- Diwan, Romesh, “Relational Wealth and the Quality of Life” (2000) 29 *Journal of Socio-Economics* 305
- Dawson, I.J., and Pearce, Robert A., *Licences Relating to the Occupation or Use of Land* (London: Butterworths, 1979)
- Dusollier, Severine, “Open source and Copyleft: Authorship reconsidered?” (2003) 26 *Columbian Journal of Law and the Arts* 283
- Dusollier, Severine, “Sharing Access to Intellectual Property Through Private Ordering”, (2007) 82 *Chicago-Kent Law Review* 1391 at 1435
- Easterbrook, Frank, “Contract and Copyright” (2005) 42 (4) *Houston Law Review* 953

- Eisenberg, Melvin A., “Why There is No Law of Relational Contracts” (2000) 94 *Northwestern University Law Review* 805
- Elkin-Koren, Niva, “What Contracts Cannot Do: The Limits of Private Ordering in Facilitating a Creative Commons” [2005] 74 *Fordham Law Review* 375
- Evans, David S. and Layne-Farrar, Anne, “Software Patents and Open Source: The Battle Over Intellectual Property Rights” (2004) 9 (10) *Virginia Journal of Law and Technology*
- Fabricius, Erich M., “Jacobsen v. Katzer: Failure of the Artistic License and Repercussions for Open Source” (2008) *North Carolina Journal of Law and Technology* 65
- Feinman, Jay M., “Relational Contract Theory in Context”, (2000) 94 *Northwestern University Law Review* 737
- Feller, Joseph and Fitzgerald, Brian, *Understanding Open Source Software Development*, (London: Addison-Wesley, 2002)
- Fisk, Catherine L., “Credit Where It’s Due: The law and Norms of Attribution”, (2006) 95 *Georgetown Law Journal* 49
- Fuller, Lon, “The Role of Contract in the Ordering Processes of Society Generally” in *The Principles of Social Order*, edited by Kenneth I. Winston (Durham, N.C.: Duke University Press, 1981)
- Free Software Consortium (Jaco Aizenman, Maureen O’Sullivan, Martin Pedersen, Pedro Rezende, Shilu Shah, Pia Smith, Jorge Villa), *Free Software Act (Draft)* (2004) 1 (4) *SCRIPT-ed* at <<http://www.law.ed.ac.uk/ahrc/script-ed/issue4/FS-Act.pdf>>
- Free Software Foundation, “Overview of the GNU System” 1996 (last update 2008) at <<http://www.gnu.org/gnu/gnu-history.html>>
- Free Software Foundation, “GPL Version 3: Background to Adoption”, 9 June 2005 <<http://www.fsf.org/news/gpl3.html>>
- Free Software Foundation, “GPL v3 Final Discussion Draft Rationale”, at <<http://gplv3.fsf.org/rationale>>
- Galli, Peter, “Rewriting GPL No Easy Task” eSeminars, 2 February 2005, eWeek, <<http://www.eweek.com/c/a/Linux-and-Open-Source/Rewriting-GPL-No-Easy-Task/>>
- Garnett, Kevin, Davies, Gillian and Harbottle, Gwilym, *Copinger and Skone James on Copyright* (London: Sweet & Maxwell, 2005, 15th Edition)
- Gates, Bill, “An Open Letter to Hobbyists”, 3 February 1976 at <http://www.digibarn.com/collections/newsletters/homebrew/V2_01/gatesletter.html>
- Ghosh, Rishab Ayer, “Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet” (1998) 3 (3) *First Monday* at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/580/501>
- Ginsburg, Jane C., “The Concept of Authorship in Comparative Copyright Law” (2003) 52 *DePaul Law Review* 1063

- Ginsburg, Jane C., "The Right to Claim Authorship in U.S. Copyright and Trademarks Law", (2004) 41 (2) *Houston Law Review* 263
- Gomulkiewicz, Robert W., "How Copyleft Uses License Rights to Succeed in the Open source Software Revolution and the Implications for the Implications for Article 2B" (1999) 36 *Houston Law Review* 179
- Gomulkiewicz, Robert W., "A First Look at General Public License 3.0", (2007) 24 *Computer and Internet Lawyer* 15
- Gomulkiewicz, Robert W., "Conditions and Covenants in License Contracts: Tales from a Test of the Artistic License" (2009) 17 *Texas IP Law Journal* 335
- Gordon, Robert W., "Macaulay, Macneil, and the Discovery of Solidarity and Power in Contract Law" (1985) *Wisconsin Law Review* 565
- Griffiths, Jonathan, "Misattribution and Misrepresentation—the Claim for Reverse Passing Off as 'Paternity' Right" [2006] 1 *I.P.Q.* 34
- Gross, Michael, "Richard Stallman: High School Misfit, Symbol of Free Software, MacArthur-certified Genius", 1999, at <<http://www.mgross.com/MoreThgsChng/interviews/stallman1.html>>
- Guadamuz, Andres, "Viral Contracts or Unenforceable Documents? Contractual Validity of Copyleft Licences" (2004) 26 (8) *E.I.P.R.* 331
- Guadamuz, Andres, "Legal Challenges to Open Source Licences" (2005) 2 (2) *SCRIPT-ed* 163
- Gudel, Paul J., "Relational Contract Theory and the Concept of Exchange", (1998) 46 *Buffalo Law Review* 763
- Hass, Douglas A., "A Gentlemen's Agreement: Assessing the GNU General Public License and its Adaptation to Linux" (2007) 6 *Chicago-Kent Journal of Intellectual Property* 213
- Henley, Mark, "Jacobsen v. Katzer and Kamind Associates—An English Legal Perspective" (2009) 1 (1) *International Free and Open Source Software Law Review*
- Heymann, Laura A., "The Birth of the Authornym: Authorship, Pseudonymity, and Trademark Law" (2005) 80 *Notre Dame Law Review* 1377
- Himanen, Pekka, *The Hacker Ethic and the Spirit of the Information Age* (NY: Random House, 2001)
- Houweling, Molly Shaffer Van, "Cultural Environmentalism and the Constructed Commons", (2007) 70 *Law and Contemporary Problems* 23
- Hyde, Lewis, *The Gift: Imagination and the Erotic Life of Property* (New York: Vintage Books, 1983)
- Irlam, Gordon and William, Ross, *Software Patents: An Industry at Risk*, 1994, at <<http://www.progfree.org/Patents/industry-at-risk.html>>

- Jaszi, Peter, "Toward a Theory of Copyright: The Metamorphoses of 'Authorship'" (1991) 2 *Duke Law Journal* 455
- Jaszi, Peter, "On the Author Effect: Contemporary Copyright and Collective Creativity" (1992) 10 *Cardozo Arts and Entertainment Law Journal* 293
- Jones, Pamela, "The GPL is a License, Not a Contract, Which is Why the Sky Isn't Falling", 14 December 2003 at
<<http://www.groklaw.net/articlebasic.php?story=20031214210634851>>
- Karjala, Dennis S., "Federal Preemption of Shrinkwrap and Online Licenses" 22 (3) *University of Dayton Law Review* 511
- Karp, James P., "A Private Property Duty of Stewardship: Changing our Land Ethic" (1993) 23 *Environmental Law* 735
- Kelty, Christopher M., *Two Bits--The Cultural Significance of Free Software*, (Durham: Duke University Press, 2008)
- Kim, Nancy S., "Clicking and Cringing", (2007) 86 *Oregon Law Review* 797
- Kretschmer, Martin, "Software as Text and Machine: The Legal Capture of Digital Innovation", 2003 (1) *The Journal of Information, Law and Technology* (JILT) at
<http://www2.warwick.ac.uk/fac/soc/law/elj/jilt/2003_1/kretschmer/>
- Kroah-Hartman, Greg, Corbet, Jonathan and McPherson, Amanda, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It: An August 2009 Update* at
<<http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf>>
- Kumar, Sapna, "Enforcing the GNU GPL" 2006 *University of Illinois Journal of Law, Technology and Policy* 1
- Kwall, Roberta Rosenthal, "The Attribution Right in the United States: Caught in the Crossfire Between Copyright and Section 43(A)" (2002) *Washington Law Review* 985
- Kwall, Roberta Rosenthal, "The Author as Steward 'For Limited Times'", (2008) *Boston University Law Review* 685
- Laddie, Hugh, Prescott, Peter, Vitoria, Mary, Speck, Adrian and Lane, Lindsay, *The Modern Law of Copyright and Designs* (London, Edinburgh & Dublin: Butterworth, 2000, 3rd edition) Vols. One & Two
- Lai, Stanley, *The Copyright Protection of Computer Software in the United Kingdom* (Oxford and Portland, Oregon: Hart Publishing, 2000)
- Lakhani, Karim R. and Wolf, Robert G., "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects", in *Perspective on Free and Open Source Software*, eds. by Feller, Fitzgerald, Hissam and Lakhani (Cambridge, Mass.: MIT Press, 2005)
- Lastowka, Greg, "The Trademark Function of Authorship", (2005) 85 *Boston University Law Review* 1172

- Leadbeater, Charles, *We-Think* (London: Profile Books, 2008)
- Leff, Arthur Allen, “Contract as Thing”, (1970) 19 (2) *American University Law Review* 131
- Lemley, Mark, “Convergence in the Law of Software Copyright”, (1995) 10 *High Technology Law Journal* 1
- Lemley, Mark, “Romantic Authorship and the Rhetoric of Property”, (1997)75 *Texas Law Review* 873
- Lemley, Mark, “Beyond Preemption: The Law and Policy of Intellectual Property Licensing” (1999) 87 (1) *California Law Review* 111
- Lemley, Mark and Shapiro, Carl, “Probabilistic Patents”, (2005) 19 (2) *Journal of Economic Perspectives* 75
- Lemley, Mark, “Terms of Use” (2006) 91 *Minnesota Law Review* 459
- Lemley, Mark, Risch, Michael, Sichelman, Ted R. and Wagner, Polk, “Life after Bilski” (2011) *Stanford Law Review* 101
- Lerner, Josh and Schankerman, Mark, *The Comingled Code: Open Source and Economic Development* (Cambridge, Mass.: MIT Press, 2010)
- Lessig, Lawrence, “Open Code and Open Societies: Values of Internet Governance” (1999) 74 *Chicago-Kent Law Review* 1405
- Lessig, Lawrence, *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity* (New York: The Penguin Press, 2004)
- Leveque, Francois and Ménière, Yann, “Copyright Versus Patents: The Open Source Software Legal Battle” (2007) 4(1) *Review of Economic Research on Copyright Issues*
- Levy, Steven, *Hackers—Heroes of the Computer Revolution* (London: Penguin Books, 1984,1994)
- Lucy, William N.R. and Mitchell, Catherine, “Replacing Private Property: The Case for Stewardship” (1996) 55 *Cambridge Law Journal* 566
- Macaulay, Stewart, “Non-Contractual Relations in Business’ (1963) 28 *American Sociological Review* 55
- Macaulay, Stewart, “The Reliance Interest and the World Outside the Law Schools’ Door”, (1991) *Wisconsin Law Review* 247
- Macaulay, Stewart, “The Real and Paper Deal: Empirical Pictures of Relationships, Complexity and the Urge for Transparent Simple Rules” (2003) 66 *Modern Law Review* 44
- Macaulay, Stewart, “Freedom from Contract: Solutions in Search of a Problem?” (2004) *Wisconsin Law Review* 777
- MacQueen, Hector, Waelde, Charlotte and Laurie, Graeme, *Contemporary Intellectual Property—Law and Policy* (Oxford: OUP, 2008)

- Madison, Michael J., “Legal-ware: Contract and Copyright in the Digital Age” (1998) 67 (3) *Fordham Law Review* 1025
- Madison, Michael J., “Legal Implications of Open-Source Software” (2001) *University of Illinois Law Review* 241
- Madison, Michael J., “Reconstructing the Software License” (2003) 35 *Loyola University Chicago Law Journal* 275
- Maier, Gregory J., “Software Protection—Integrating Patent, Copyright and Trade Secret Law” (1987) 69 *Journal of Patent and Trademark Office Society* 151
- Ronald J. Mann, “Do Patents Facilitate Financing in the Software Industry?” (2005) 83 (4) *Texas Law Review* 961
- Macneil, I.R., “The Many Futures of Contracts” (1973-74) 47 *South California Law Review* 692
- Macneil, I.R., “Restatement (Second) of Contracts and Presentation” (1974) 60 (4) *Virginia Law Review* 589
- Macneil, I.R., “Contracts: Adjustment of Long-Term Economic Relations under Classical, Neoclassical and Relational Contract Law” (1978) 72 *Northwestern University Law Review* 854
- Macneil, I.R., *The New Social Contract—An Inquiry into Modern Contractual Relations* (New Haven and London: Yale University Press, 1980)
- Macneil, I.R., “Economic Analysis of Contractual Relations: Its Shortfalls and the Need for a Rich ‘Classificatory Apparatus’ ”, (1981) 75 *Northwestern University Law Review* 1018
- Macneil, I.R., “Bureaucracy and Contracts of Adhesion” (1984) 22 *Osgoode Hall Law Journal* 5
- Macneil, I.R., “Relational Contract: What We Do and Do not Know” (1985) 3 *Wisconsin Law Review* 483
- Macneil, I.R., “Exchange Revisited: Individual Utility and Social Solidarity” (1986) 96 (3) *Ethics* 567
- Macneil, I.R., “Contracting Worlds and Essential Contract Theory” (2000) 9 *Social and Legal Studies* 431
- Macneil, I.R., “Reflection on Relational Contract Theory after a Neo-classical Seminar”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. By David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003)
- Marrella, Fabrizio & Yoo, Christopher S., “Is Open Source Software the New Lex Mercatoria?” (2007) 47 (4) *Virginia Journal of International Law*
- McGowan, David, “Legal Implications of Open-Source Software” (2001) *University of Illinois Law Review* 241

- McLaughlin, Nancy A., “Rethinking the Perpetual Nature of Conservation Easements”, (2005) 29 *Harvard Environmental Law Review* 421
- Merges, Robert, “The End of Friction? Property Rights and Contract in the ‘Newtonian’ World of On-line Commerce (1997) 12 *Berkeley Technology Law Journal* 115
- Merges, Robert, “Software and Patent Scope: A Report from the Middle Innings” (2007) 85 *Texas Law Review* 1627
- Merges, Robert, “The Concept of Property in the Digital Era” [2008] 45 (4) *Houston Law Review* 1239
- Merrill, Thomas W. and Smith, Henry E., “The Property/Contract Interface”, (2001) 101 (4) *Columbia Law Review* 773
- Miller, Arthur, “Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU” (1993) 106 *Harvard Law Review* 977
- Moglen, Eben, “Anarchism Triumphant: Free Software and the Death of Copyright”, (1999) 4 (8) *First Monday*, at <http://www.firstmonday.org/issues/issue4_8/moglen/>
- Moglen, Eben, *The dotCommunist Manifesto*, January 2003, <<http://emoglen.law.columbia.edu/publications/dcm.html>>
- Moglen, Eben, “Freeing the Mind: Free Software and The Death of Proprietary Culture”, 29 June 2003, <<http://moglen.law.columbia.edu/publications/maine-speech.html>>
- Mouzas, Stefanos and Furmston, Michael “From Contract to Umbrella Agreement” (2008) 67(1) *Cambridge Law Journal* 37
- Mulcahy, Linda, and Andrews, Cathy, “Baird Textile Holdings v Marks & Spencer Plc” in *Feminist Judgements—From Theory to Practice* (Oxford and Portland, Oregon: Hart, 2010)
- Nadan, Christian H., “Open Source Licensing: Virus or Virtue?” [2002] *Texas Intellectual Property Law Journal* 349
- Narodick, Benjamin I., “Smothered by Judicial Love: How *Jacobsen v. Katzer* Could Bring Open Source Software Development to a Standstill” (2010) 16 *Boston University Journal of Science and Technology Law* 264
- Netscape, “Netscape Announces Plans to Make Next-Generation Communicator Source Code Available Free on the Net” 1998 at <<http://wp.netscape.com/newsref/pr/newsrelease558.html>>
- Nimmer, Raymond T., “Breaking Barrier: The Relationship between Contract and Intellectual Property Law”, (1998) 13 *Berkeley Technology Law Journal* 827
- Nimmer, David, Brown, Elliot and Frischling, Gary N., “The Metamorphosis of Contract into Expand”, (1999) 87 (1) *California Law Review* 17
- O’Reilly, Tim, “Lessons from Open-Source Software Development”, (1999) 42 (4) *Communications of the ACM* 33

O'Reilly, Tim, "The Open Source Paradigm Shift", in *Open Sources 2.0*, edited by Chris DiBona, Danese Cooper, and Mark Stone (Sebastopol, CA: O'Reilly, 2006)

Passmore, John, *Man's Responsibility for Nature—Ecological Problems and Western Traditions* (London: Duckworth, 1974)

Patterson, Chip, "Copyright Misuse and Modified Copyleft: New Solutions to the Challenges of Internet Standardization", (2000) 98 *Michigan Law Review* 1351

Perens, Bruce, "Open Source Definition" in *Open Sources: Voices from the Open Source Revolution* eds. by Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O'Reilly & Associates, 1999)

Pila, Justine, "Dispute over the Meaning of 'Invention' in Art. 52(2) EPC—The Patentability of Computer-Implemented Inventions in Europe" (2005) 36 *IIC* (2) 173

Pila, Justine, "Software Patents, Separation of Powers, and Failed Syllogisms: A Cornucopia from the Enlarged Board of Appeal of the European Patent Office, (2010) *Oxford Legal Research Paper Series*, Paper No 48/2010

Posner, Eric, "ProCD v Zeidenberg and Cognitive Overload in Contractual Bargaining" (2010) 77 *The University of Chicago Law Review* 1181

Post, Deborah W., "Dismantling Democracy: Common Sense and the Contract Jurisprudence of Frank Easterbrook", (2000) 16 *Touro Law Review* 1205

Puckette, Miller, "Who Owns Our Software—A First-person Case Study", 2004 *ISEA Online Proceedings*, available at <<http://crca.ucsd.edu/~msp/Publications/isea-reprint.pdf>>

Radin, Margaret Jane, "Humans, Computers, and Binding Commitment" (1999) 75 *Indiana Law Journal* 1125

Radin, Margaret Jane, "Boilerplate Today, The Rise of Modularity and the Waning of Consent" (2006) 104 *Michigan Law Review* 1223

Raymond, Eric (editor), *The New Hacker's Dictionary* (or "Jargon File") at <<http://www.catb.org/~esr/jargon/html/H/hacker.html>>

Raymond, Eric, *The Magic Cauldron*, 1999 at <<http://www.catb.org/~esr/writings/magic-cauldron/>>

Raymond, Eric, *The Cathedral and the Bazaar*, 2000, version 3.0 at <<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>>

Raymond, Eric, *How to Become a Hacker*, 2001, at <<http://www.catb.org/~esr/faqs/hacker-howto.html>>

Raymond, Eric, *Homesteading the Noosphere*, 2002 at <<http://www.catb.org/~esr/writings/homesteading/homesteading/>>

Raymond, Eric and Raymond, Catherine Olanich, *Licensing HOWTO*, 9 November 2002, at <<http://catb.org/~esr/Licensing-HOWTO.html>>

- Rose, Mark, *Authors and Owners—The Invention of Copyright*, (Cambridge, Mass. & London: Harvard University Press, 1993)
- Rosen, Lawrence, *Open Source Licensing—Software Freedom and Intellectual Property Law*, (Upper Saddle River, NJ: Prentice Hall PTR, 2005)
- Rowland, Dane and Campbell, Andrew, “Supply of Software: Copyright and Contract Issues” (2002) 10 (1) *International Journal of law and Information Technology* 23
- Rychlicki, Tomasz, “GPLv3: New Software Licence and New Axiology of Intellectual Property Law” (2008) 30 (6) *European Intellectual Property Review* 232
- Samuelson, Pamela, Davis, Randall, Kapor, Mitchell D., Reichman, J. H., “A Manifesto concerning the Legal Protection of Computer Programs”, (1994) 94 (8) *Columbia Law Review* 2308
- Samuelson, Pamela, “The Quest for Enabling Metaphors for Law and Lawyering in the Information Age” (1996) 94 (6) *Michigan Law Review* 2029
- Saunders, David, *Authorship and Copyright*, (London: Routledge, 1992)
- Schwarzenbach, Sibyl, “Locke’s Two Conceptions of Property” (1988) 14 (2) *Social Theory and Practice* 141
- Sennett, Richard, *The Craftsman* (New Haven & London: Yale University Press, 2008)
- Shankland, Stephen, “Defender of the GPL”, CNET News.com, 19 January 2006 at <http://news.cnet.com/Defender-of-the-GPL/2008-1082_3-6028495.html>
- Shemtov, Noam, “The Characteristics of Technical Character and the Ongoing Saga in the EPO and the English Courts” (2009) 4(7) *Journal of Intellectual Property Law & Practice* 506
- Shemtov, Noam, “Software Patents and Open Source Models in Europe: Does the FOSS Community Need to Worry about Current Attitudes at the EPO?” (2010) 2 (2) *International Free and Open Source Software Law Review* 151
- Shils, Edward, “Henry Sumner Maine in the Tradition of the Analysis of Society”, in *The Victorian Achievement of Sir Henry Maine: A Centennial Reappraisal*, ed. By Alan Diamond (Cambridge: CUP, 2001)
- Shindler, Andrew, “Derogation from Grant in Copyright Law” (1986) 49 (4) *Modern Law Review* 513
- Slawson, David, W., “Standard Form Contracts and Democratic Control of Lawmaking Power” (1971) 84 (3) *Harvard Law Review* 529
- Smith, Roger, *Property Law* (Essex, England: Pearson Education Limited, 2003, 4th Edition)
- Software Freedom Law Center, “Originality Requirements under U.S. and E.U. Copyright Law”, 27 September 2007, at <<http://www.softwarefreedom.org/resources/2007/originality-requirements.html>>

Software Freedom Law Center (Richard Fontana et. al.), *A Legal Issues Primer for Open Source and Free Software Projects*, 3 March 2008, at <<http://www.softwarefreedom.org/resources/2008/foss-primer.html>>

Stallman, Richard, “Initial Announcement”, 1983, at <<http://www.gnu.org/gnu/initial-announcement.html>>

Stallman, Richard, *The GNU Manifesto*, 1985, at <<http://www.gnu.org/gnu/manifesto.html>>

Stallman, Richard, “GNU Emacs availability information”, 3 July 1985 at <<http://mirror.libre.fm/MIT/gnu/emacs-16.56/etc/DISTRIB>>

Stallman, Richard, “Why Software Should Be Free”, 1991, at <<http://www.gnu.org/philosophy/shouldbefree.html>>

Stallman, Richard, “Why Software Should Not Have Owners”, 1994, at <<http://www.gnu.org/philosophy/why-free.html>>

Stallman, Richard, “The Free Software Definition”, 1996, at <<http://www.gnu.org/philosophy/free-sw.html>>

Stallman, Richard, “Patent Reform Is Not Enough”, 1996, at <<http://www.gnu.org/philosophy/patent-reform-is-not-enough.html>>

Stallman, Richard, “The GNU Operating System and the Free Software Movement” in *Open Sources: Voices from the Open Source Revolution* eds. by Chris DiBona, Sam Ockman & Mark Stone (Sebastopol, O'Reilly & Associates, 1999)

Stallman, Richard, “Why We Must Fight UCITA”, 31 January 2000 at <http://w2.eff.org/IP/UCITA_UCC2B/20000131_fight_ucita_stallman_paper.html>

Stallman, Richard, “Software Patents—Obstacles to Software Development”, script of a speech delivered at the University of Cambridge Computer Lab, 25 March 2002, at <<http://www.cl.cam.ac.uk/~mgk25/stallman-patents.html>>

Stallman, Richard, “On Hacking”, 2002, at <<http://stallman.org/articles/on-hacking.html>>

Stallman, Richard, “Fighting Software Patents—Singly and Together”, 2004, at <<http://www.gnu.org/philosophy/fighting-software-patents.html>>

Stallman, Richard, “The Dangers of Software Patents”, 24 May 2004, a talk delivered at the University of Dublin, Trinity College, organised by Irish Free Software Organisation, transcript by Glenn Strong, Malcolm Tyrrell, Aidan Delaney and Ciaran O’Riordan at <<http://www.ifso.ie/documents/rms-2004-05-24.html>>

Stallman, Richard, “Did You Say ‘Intellectual Property’? It’s a Seductive Mirage”, 2004 at <<http://www.gnu.org/philosophy/not-ipr.html>>

Stallman, Richard, “Why ‘Free Software’ is Better than ‘Open Source’?” 2005 at <<http://www.gnu.org/philosophy/free-software-for-freedom.html>>

Stallman, Richard, *GNU Emacs Manual* (Boston, MA: Free Software Foundation, 2010, 16th Edition)

- Stallman, Richard, “What’s in a Name?” at <<http://www.gnu.org/gnu/why-gnu-linux.html>>
- Stokes, Simon, “The Development of UK Software Copyright Law: from John Richardson Computers to Navitaire” (2005) 11 (4) *Computer and Telecommunications Law Review* 129
- Tapscott, Don and Williams, Anthony D., *Wikinomics* (London: Portfolio, 2006)
- Titmuss, Richard M., *The Gift Relationship—From Human Blood to Social Policy*, eds. by Ann Oakley and John Ashton, (NY: The New Press, 1997; Originally published in 1970)
- Torvalds, Linus, “What Make Hackers Tick? a.k.a. Linus’s Law” as the prologue to *The Hacker Ethic and the Spirit of the Information Age*, by Pekka Himanen, (NY: Random House, 2001)
- Treitel, Guenter, *The Law of Contract*, (London: Sweet and Maxwell, 2003, 11th Edition)
- Tushnet, Rebecca, “Naming Rights: Attribution and Law” (2007) 3 *Utah Law Review* 781
- Vaidhyathan, Siva, “The Anarchist in the Coffee House: A Brief Consideration of Local Culture, The Free Culture Movement, and Prospects for a Global Public Sphere”, (2007) 70 (2) *Law and Contemporary Problems* 205
- Wacha, Jason B., “Taking the Case: Is the GPL Enforceable” (2005) 21 *Santa Clara Computer and High Technology Law Journal* 451
- Wagner, R. Polk “Information Wants to Be Free—Intellectual Property and the Mythologies of Control, (2003) 102 *Columbia Law Review* 995, in *Intellectual Property: Critical Concepts in Law*, edited by David Vaver (Oxford: Routledge, 2006)
- Wayner, Peter, *Free for All—How Linux and the Free Software Movement Undercut the High-Tech Titans* (HarperBusiness, 2000) also available at <http://www.jus.uio.no/sisu/free_for_all.peter_wayner/>
- Weber, Steven, *The Success of Open Source* (Cambridge, Mass.: Harvard Uni. Press, 2004)
- Williams, Sam, *Free as in Freedom--Richard Stallman's Crusade for Free Software*, O’Reilly, 2002 at <<http://www.oreilly.com/openbook/freedom/>>
- Wightman, John, “Beyond Custom: Contract, Contexts, and the Recognition of Implicit Understandings”, in *Implicit Dimensions of Contract—Discrete, Relational, and Network Contracts*, eds. By David Campbell, Hugh Collins and John Wightman (Oxford and Portland, Oregon: Hart Publishing, 2003)
- Whitford, William C., “Ian Macneil’s Contribution to Contracts Scholarship”, (1985) *Wisconsin Law Review* 545
- Woodmansee, Martha, “The Genius and Copyright” in *The Author, Art, and the Market—Reading the History of Aesthetics* (NY: Columbia University Press, 1994) originally published in (1984)17 *Eighteenth-Century Studies* 425, titled “The Genius and Copyright: Economic and Legal Conditions of the Emergence of the ‘Author’”
- Woodmansee, Martha, “On the Author Effect: Recovering Collectivity” (1992) 10 *Cardozo Arts and Entertainment Law Journal* 279

Zimmerman, Diane Leenheer, "Authorship without Ownership: Reconsideration Incentives in a Digital Age" (2003) *52 DePaul Law Review* 1121

Appendix (A): Development of “Intellectual Property” and FOSS: A Timeline

	Copyright	Patent	Trademark	Miscellaneous
1972		<i>Gottschalk v. Benson</i> (US SC)		
1973		European Patent Convention		
1978	US CONTU recommendation			
1980	US Congress amended its Copyright Act to expressly cover software			
1981		<i>Diamond v. Diehr</i> (US SC)		
1983				Emacs dispute between Stallman and Gosling
1984				Steven Levy documented the Hacker Ethic
1985				Emacs GPL (first copyleft licence)
1986	<i>Whelan v. Jaslow</i> (3 rd Cir.)			
1987		<i>Vicom/Computer- related invention</i> (EPO)		
1988	UK CDPA (expressly recognising copyright subsistence in “software”)			
1989				-Stallman’s Anti- Lotus Litigation Protest -GPL 1.0
1990	<i>Lotus v. Paperback</i> (first Lotus case)			
1991	EU Software Directive			
1992	<i>Computer Associates v. Altai</i> (2 nd Cir.)			GPL 2.0
1996	* <i>ProCD v. Zeidenberg</i> (7 th Cir.)			
1998		<i>State Street Bank v. Signature Financial Group</i> (CAFC)		Open Source Initiative (OSI) founded

2001			<i>Planetary Motion v. Techsplosion</i>	
2002		<i>PBS Partnership/Pension Benefit System (EPO)</i>		
2004		<i>Hitachi/Auction Method (EPO)</i>		
2006		<i>Microsoft/Clipboard Form I&II (EPO)</i>		Open Source As Prior Art (OSAPA) launched
2007		<i>Aerotel Ltd. v. Telco Holdings Ltd. (UK CA)</i>		GPL 3.0
2008	<i>Jacobsen v. Katzer (CAFC)</i>	<i>Bilski v. Kappor (CAFC)</i>		
2010		<i>Bilski v. Kappor (US SC)</i>		

(*Author's Note: The *ProCD* case is not purely a copyright case. More importantly, it deals with an intersectional area covering both copyright and contract. See Chapter 4 of this dissertation for a more detailed analysis.)

Appendix (B): GNU Emacs General Public License (1985)

*originally published in 1985, clarified 11 February 1988***

The license agreements of most software companies keep you at the mercy of those companies. By contrast, our general public license is intended to give everyone the right to share GNU Emacs. To make sure that you get the rights we want you to have, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. Hence this license agreement.

Specifically, we want to make sure that you have the right to give away copies of Emacs, that you receive source code or else can get it if you want it, that you can change Emacs or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of Emacs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for GNU Emacs. If Emacs is modified by someone else and passed on, we want its recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

Therefore we (Richard Stallman and the Free Software Foundation, Inc.): make the following terms which say what you must do to be allowed to distribute or change GNU Emacs.

Copying Policies 1. You may copy and distribute verbatim copies of GNU Emacs source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each file a valid copyright notice "Copyright 1988 Free Software Foundation, Inc." (or with whatever year is appropriate); keep intact the notices on all files that refer to this License Agreement and to the absence of any warranty; and give any other recipients of the GNU Emacs program a copy of this License Agreement along with the program. You may charge a distribution fee for the physical act of transferring a copy.

2. You may modify your copy or copies of GNU Emacs source code or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:

- cause the modified files to carry prominent notices stating who last changed such files and the date of any change; and

- cause the whole of any work that you distribute or publish, that in whole or in part contains or is a derivative of GNU Emacs or any part thereof, to be licensed at no charge to all third parties on terms identical to those contained in this License Agreement (except that you may choose to grant more extensive warranty protection to some or all third parties, at your option).

- if the modified program serves as a text editor, cause it, when started running in the simplest and usual way, to print an announcement including a valid copyright notice "Copyright 1988 Free Software Foundation, Inc." (or with the year that is appropriate), saying that there is no warranty (or else, saying that you provide a warranty) and that users

may redistribute the program under these conditions, and telling the user how to view a copy of this License Agreement.

- You may charge a distribution fee for the physical act of transfer ring a copy, and you may at your option offer warranty protection in exchange for a fee.

Mere aggregation of another unrelated program with this program (or its derivative) on a volume of a storage or distribution medium does not bring the other program under the scope of these terms.

3. You may copy and distribute GNU Emacs (or a portion or derivative of it, under Paragraph 2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:

- accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,

- accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal shipping charge) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,

- accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for non commercial distribution and only if you received the program in object code or executable form alone.)

For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs.

4. You may not copy, sub license, distribute or transfer GNU Emacs except as expressly provided under this License Agreement. Any attempt otherwise to copy, sub license, distribute or transfer GNU Emacs is void and your rights to use GNU Emacs under this License agreement shall be automatically terminated. However, parties who have received computer software programs from you with this License Agreement will not have their licenses terminated so long as such parties remain in full compliance.

5. If you wish to incorporate parts of GNU Emacs into other free programs whose distribution conditions are different, write to the Free Software Foundation. We have not yet worked out a simple rule that can be stated here, but we will often permit this. We will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software.

Your comments and suggestions about our licensing policies and our software are welcome! Please contact the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139.

NO WARRANTY

BECAUSE GNU EMACS IS LICENSED FREE OF CHARGE, WE PROVIDE ABSOLUTELY NO WARRANTY, TO THE EXTENT PERMITTED BY APPLICABLE STATE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING, FREE SOFTWARE FOUNDATION, INC, RICHARD M. STALLMAN AND/OR OTHER PARTIES PROVIDE GNU EMACS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A

PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE GNU EMACS PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW WILL FREE SOFTWARE FOUNDATION, INC., RICHARD M. STALLMAN, AND/OR ANY OTHER PARTY WHO MAY MODIFY AND REDISTRIBUTE GNU EMACS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH PROGRAMS NOT DISTRIBUTED BY FREE SOFTWARE FOUNDATION, INC.) THE PROGRAM, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

(**Author's Note: This is the very first copyleft licence written by Stallman as his response to the dispute with Gosling over a version of Emacs editor from 1983 to 1985. It is followed by the three generic versions of GNU General Public License respectively published in 1989, 1992 and 2007. For the history of the GPL, see Chapters 2 and 3 of this dissertation for more detail.)