

USE OF KNOWLEDGE COMMONS IN OPEN INNOVATION SYSTEMS: THE
CASE OF FREE AND OPEN SOURCE SOFTWARE

by

Justin M. Novak
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Public Policy

Committee:

_____	David Hart, Chair
_____	Jeremy Mayer
_____	Phillip Auerswald
	David Audrestch, External Reader
_____	Sita N. Slavov, Program Director
_____	Mark J. Rozell, Dean
Date: _____	Fall Semester 2016 George Mason University Fairfax, VA

Use of Knowledge Commons in Open Innovation Systems: The Case of Free and Open
Source Software

A Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

by

Justin M. Novak
Bachelor of Science
University of Pittsburgh, 2006

Director: David Hart, Professor
Department of Public Policy

Fall Semester 2016
George Mason University
Fairfax, VA



This work is licensed under a [creative commons attribution-noncommercial 3.0 unported license](https://creativecommons.org/licenses/by-nc/3.0/).

DEDICATION

This dissertation is dedicated to my Parents. Mom, the teacher who gave me a lifelong love of learning and Dad, of whom some of my earliest memories are him going to night school to get his degree. You taught me the value of hard work and the value of education.

ACKNOWLEDGEMENTS

There are many people I would like to thank, beginning with my committee, Drs. Hart, Mayer, and Auerswald, particularly Dr. David Hart for agreeing to chair and lead my committee. Also, thank you to Dr. David Audrestch for agreeing to serve as the external reader for this dissertation. I would also like to thank Dr. Sonia Ketkar for serving on an earlier version of my committee and Dr. John Gordon for his input on my research. I have also been fortunate to have understanding and accommodating employers who allowed me the leeway and the time to dedicate towards this pursuit.

The greatest measure of gratitude is reserved for my wife, Kristin. You have been more than just a loving spouse throughout this journey. You have gone above and beyond to be an editor, fact checker, idea incubator, sounding board, and consultant. All in addition to providing incredible support in every way. I could not have done this without you, thank you.

TABLE OF CONTENTS

	Page
List of Tables	vii
List of Figures	viii
List of Abbreviations	ix
Abstract	x
Chapter One: Introduction	1
Theoretical Background and Significance	1
Chapter Two: Review of the Literature	7
Introduction	7
Open Innovation	7
The Importance of Commons in Innovation	9
The Software Industry	13
The Free and Open Source Software Community: Powering Open Innovation?	17
Chapter Three: Understanding Innovation in Free and Open Source Systems: A Case Study Methodology	26
Introduction	26
The Challenge of Measuring Innovation	26
Research Question	31
Hypotheses	32
Case Selection	38
Variables	41
Study Design	47
Chapter Four: Innovation in the Snort Community	53
Introduction	53
The Snort Open Source Community	54
Direct Input: Community-Driven Innovation	58
Shifts in Community Behavior and Makeup	63
Knowledge Absorption: Community Contributions to Internal Innovation	70
Talent Acquisition by the Firm	74
Conclusion	76

Chapter Five: Open Source, Innovation, and Security	78
Introduction	78
Software Security	78
Open Source-Fueled Security and Innovation	80
Security Fixes as Innovation	83
Characterizing Security Contributions	90
Security and External Knowledge	94
Chapter Six: Governance and Open Innovation in Snort.....	97
Introduction	97
Models of Governance for Software Innovation and Open Source	98
Governance and the FOSS Approach.....	100
The Snort Project – A Model of Governance in Open Source.....	103
Snort Project Governance.....	103
Changes over Time – the Shifting Sands of FOSS Communities	112
Chapter Seven: Conclusions and Implications for Policy.....	120
Is Open Source a Path to Greater Innovation?	120
Policy Implications.....	125
Appendix I – Bug Contributor Affiliations.....	131
Appendix II – Security ‘Super Contributor’ Affiliations.....	134
Appendix III – SDLC Models.....	136
References	138

LIST OF TABLES

Table	Page
Table 1: Theoretical concepts: Tragedy of the commons and anti-commons	12
Table 2: Description of empirical measurement	37
Table 3: Inter-coder reliability check.....	50
Table 4: Inter-coder reliability check – Referee 1	51
Table 5: Inter-coder reliability check – Referee 2	51
Table 6: Summary statistics for Snort-devel email listserv archive	55
Table 7: Summary statistics for bug contributors	87
Table 8: Behaviors of all bug contributors	92
Table 9: Behaviors of fly-by vs. super contributors	93

LIST OF FIGURES

Figure	Page
Figure 1: Snort-devel listserv conversation 1	60
Figure 2: Snort-devel listserv conversation 2	61
Figure 3: Snort-devel listserv conversation 3	62
Figure 4: Predicted Snort FOSS community activity	64
Figure 5: Actual Snort FOSS community activity	65
Figure 6: Bug submissions to the email listserv over time	67
Figure 7: Feature submissions to the email listserv over time	68
Figure 8: Code submissions to the email listserv over time	69
Figure 9: Snort-devel listserv conversation 4	70
Figure 10: Snort-devel listserv conversation 5	72
Figure 11: Stages of the SDLC (BearingPoint, 2012)	99
Figure 12: FOSS governance within the SDLC (BearingPoint, 2012)	102
Figure 13: Sourcefire/Cisco open source project governance structure	104
Figure 14: Snort-devel listserv conversation 6	108
Figure 15: Snort-devel listserv conversation 7	109
Figure 16: Activity surrounding version release dates	111
Figure 17: Anticipated shift in contribution types	114
Figure 18: Changes in firm preference toward community	116

LIST OF ABBREVIATIONS

Intellectual Property.....	IP
Intellectual Property Rights	IPR
Software Development Life Cycle.....	SDLC
Free and Open Source Software	FOSS
Comprehensive National Cybersecurity Initiative.....	CNCI
Internet Relay Chat	IRC

ABSTRACT

USE OF KNOWLEDGE COMMONS IN OPEN INNOVATION SYSTEMS: THE CASE OF FREE AND OPEN SOURCE SOFTWARE

Justin M. Novak, Ph.D.

George Mason University, 2016

Dissertation Director: Dr. David Hart

The idea that innovation is essential to economic growth has become more widely accepted in policy circles in recent years and is being increasingly adopted in practice by policymakers. The literature surrounding innovation is becoming clearer about its effects on the creation of new ideas and products. However, many unknowns remain regarding the causes of innovation, including the extent to which openness aids innovation and how knowledge should be treated within innovation systems. This study examines innovation inputs in the software industry, where the dominant open innovation scheme is Free and Open Source Software (FOSS). Using a mixed methods case study, it explores how knowledge commons allow external knowledge to be introduced into an open innovation system, which in turn provides access to resources that would not be otherwise available. The results

demonstrate the important roles of governance, community, and collaboration and have broader implications for innovation systems.

CHAPTER ONE: INTRODUCTION

Theoretical Background and Significance

Policy makers widely consider innovation to be a crucial means of promoting economic prosperity. The study of such innovation-driven growth dates back more than a century and has been conducted by a host of prominent scholars (e.g., Kline, 1985; Teece, 1986; von Hippel 2005, 2007; Jaffe, 2011). As the importance of innovation has become more evident, much of the relevant research and discussion has focused on establishing which policies best foster an innovative economy. One area that has received extensive attention is the treatment and protection of intellectual property (IP). The system of protecting creative and scientific works through a regime of patent, trademark, and copyright was indeed created for the very purpose of promoting new ideas and products – language in the U.S. Constitution says as much. However, more recent research has suggested that this regime may not be working as intended; in fact, the structure of IP laws may in some cases even prove a barrier to innovation (Mansfield, 1986; Heller and Eisenberg, 1998). Alternative theories of innovation have sought to explain why this may happen and how the problem could be avoided. Among those theories, the idea of open innovation stands out as a paradigm with the potential to explain a range of innovation-related issues, including how IP can be treated in order to best harness the potential power of innovation systems.

Open innovation is a model that emphasizes the use of external knowledge. Henry Chesborough (2003), whose ideas have been critical in expanding the collective academic

understanding of open innovation, points out that in the open model, this external knowledge can be disseminated throughout the system by any method – including the licensing or selling of IP rights (IPR) via the aforementioned regulatory system of patent, copyright, etc. As a result, it is important to emphasize that open innovation does not discount the use of IPR. However, there is reason to suspect that the presence of IPR is not a necessary – or even desirable – condition to foster innovation, and that the use of IP protections may in fact be harmful to innovation (even in the case of open innovation). According to this line of thought, the transaction costs and other barriers associated with IPR harm innovation systems by stifling knowledge flows, which suggests that open innovation is not enough and that the specific treatment of IPR is a critical piece of the puzzle. The important role of these factors – which together form the *governance* of a particular project – are discussed further in Chapter Six.

This study examines the innovation dynamic in open innovation schemes that do not take advantage of strong IPR protections. In the software industry, the dominant open model of this type is Free and Open Source Software (FOSS): software programs or projects for which the code is made available publicly and contributions from a larger community are integrated into that code. Put another way, FOSS is simply software whose source code (namely, the code that constitutes the core of the program) is shared freely with others for their own use. It should be noted that this does not *necessarily* mean that the software is available free of charge (although it could); it instead refers to a freedom of sharing information. Richard Stallman, who is often considered the father of the FOSS movement, frequently notes that we should think ‘free’ as in ‘free speech,’ not ‘free’ as in ‘free beer’ (Williams and Stallman, 2010). Moreover, FOSS explicitly calls

for the loosening or outright elimination of IPR usage. In the FOSS community, the emphasis on free and ready access to knowledge (in place of IPR protections) is generally understood to lead to the organic growth of a *knowledge commons*. According to Elinor Ostrom (1990, 1995), a knowledge commons is any shared body of information that the many members of a community both draw from and contribute to. The commons, as envisioned by Ostrom, provides an opportunity for the community to self-govern and decide how information is to be shared and used. It has been suggested (e.g., Raymond, 1998; Williams and Stallman, 2010) that using the FOSS knowledge commons may enable software firms to achieve greater rates of innovation than they may otherwise, even if the particular type of open innovation applied does not take advantage of the FOSS commons and instead relies on licensing and other more traditional means of acquiring IP. This dissertation focuses only on the model that does use the FOSS commons.

The primary concepts of this study are thus innovation and the use of the FOSS model, as indicated by the absorption of external knowledge into the innovation system. The results have the potential to add significant insights to the growing body of literature on open innovation and knowledge commons, as well as to enhance the understanding of one of the key components of innovation – namely the treatment of IP – and how that treatment affects the flow of knowledge that is critical to innovation. Most immediately, the results may suggest to software firms and business leaders whether models such as FOSS present an opportunity to gain an innovation advantage as compared to rival approaches.

More importantly, however, this research has significant policy implications as the world economy continues to move into and through the information age. Questions of innovation are becoming more critical for achieving economic growth, while the ways in which we handle knowledge in an information economy are simultaneously being re-thought. Understanding these issues is important to governments around the world. The research conducted thus far on FOSS as it relates to science, technology, and innovation policy has focused on describing how policy may shape a FOSS commons. It has not, however, demonstrated why the innovation outcomes of open source products may be superior to those of non-open source products. It is this area to which the research described below adds, by demonstrating how the FOSS commons provides both greater amounts and new forms of external knowledge that might be used to improve innovation outcomes.

This study therefore contributes to the existing body of research on innovation (specifically open innovation), open source, and knowledge commons in a number of ways, based on a case study that allows for a unique exploration of these issues. In particular, this dissertation:

- Adds empirical knowledge related to an oft-postulated idea that open innovation in general and innovation in the FOSS community in particular are superior due to the knowledge from the community. Popularized by writings such as those of Williams and Stallman (2010) and Raymond (1998), this theory has gained widespread acclaim and a great deal of research has been undertaken to explore the root causes of the connection between open source

and innovation. This study adds to the body of knowledge by describing the size, scope, and nature of the communities' knowledge contributions;

- Describes the treatment of external knowledge from the FOSS commons in a holistic fashion, exploring issues ranging from community building, governance, and specialization, to types of contributions and the backgrounds of those contributing to innovation. Due to its longitudinal nature, the case study also provides an opportunity to explore how all of these issues shift and change over time, as well as what these changes mean for innovation;
- Offers a new dataset that provides a novel perspective on the issue at hand. This dataset aids in the previously mentioned empirical test of open sources' effects on innovation by qualitatively measuring discrete inputs from the FOSS commons to the project. It also offers opportunities for future research on open innovation, FOSS, and potentially other topics as well.

The remainder of this document is organized into six chapters as follows. Chapter Two, which presents a detailed review of the literature, provides the reader background on the software industry, the FOSS sub-sector of that industry, and the importance of this new form of organization. It also discusses the knowledge commons and open innovation more generally, i.e., outside of the software industry, to provide background, context, and validity. Chapter Three offers a discussion of the methodological approach used in this dissertation, beginning with the research question, hypotheses, and theory. It then

transitions into a review of the specific research, from variables used to case selection for the case study. This chapter forms the basis for the analysis. Chapter Four is the first of three core chapters that present the results of the research described in Chapter Three, along with the analysis of the empirical work of this study. The topic addressed by this first empirical chapter is innovation in the FOSS community, where two methods of enhanced innovation via the FOSS community are apparent: direct contributions of knowledge by community members and indirect absorption of knowledge by a firm. Each of these methods is discussed in turn in Chapter Four. Chapter Five then continues the discussion with a special focus on the handling of software security bugs in the FOSS community. While this issue is unique to software, it nonetheless has implications for other open innovation communities given that it describes a unique way in which the FOSS approach builds communities of interest and specialization, as well as binds together a knowledge commons community. Chapter Six, the final core empirical chapter, offers insight into the all-important role that the governance of the FOSS commons plays in maximizing the innovative utility of that commons. A full discussion of governance highlights a longitudinal look at how governance has changed in the case studied and how these changes have affected innovation. Chapter Seven concludes with a review of the insights, implications, and additions to the body of knowledge that are offered by this dissertation. Several important policy implications are also discussed in this final chapter.

CHAPTER TWO: REVIEW OF THE LITERATURE

Introduction

The research below utilizes the software industry – specifically the FOSS subset – as a case study to demonstrate the impact of the commons on innovation. The literature review therefore necessarily considers both the software industry in general and the FOSS sector in particular. However, because this dissertation’s main research questions focus on addressing complex questions of innovation, openness, and knowledge commons, each of these topics is also discussed in turn within this literature review. For the purpose of tying these many issues together, however, the most logical place to begin is with a brief overview of the history of the study of open innovation and related concepts. This is followed by a more in-depth discussion of the role of the knowledge commons within this relationship. The software industry is then reviewed, which leads naturally to an introduction to the FOSS sector and its value as an open innovation case study.

Open Innovation

In a departure from classical and neo-classical economics and their emphasis on capital and labor, supply and demand, and a maximization of utility and profit, innovation economics suggests that the keys to growth are entrepreneurship and other disruptive forces that displace old orders and ways of doing business. Joseph Schumpeter (1942), perhaps the most well-known innovation economist, made famous the concept of ‘creative destruction’ – namely the new ideas and technological changes introduced by innovators that are capable of causing a sometimes radical alteration of economic growth. Acknowledging that innovation is critical to economic growth leads to the basis for this

study, which is exploring how innovation occurs in the open paradigm (as exemplified by the software industry, specifically FOSS) and what it is that sets firms that use open models apart from those that do not.

The previously discussed work of Henry Chesborough (2003, 2006) introduces the paradigm of ‘open innovation,’ which places great emphasis on the treatment of external knowledge (i.e., the ideas and IP that reside outside a firm). The argument here is that in a world where knowledge is increasingly distributed, it is difficult for firms to innovate using only internal knowledge. When firms grab hold of knowledge that traditionally rests outside of them, however, innovation increases and they thrive and grow. Since Chesborough, a great deal of research has demonstrated strong support for the idea of open innovation. The works of Rajala et al. (2012) and Lorenzi and Rossi (2008) have specifically explored the phenomenon in software firms, and many others have studied its effects in a variety of other industries. However, beyond proving that open innovation is occurring, this dissertation seeks to begin describing *how* and *why* it is successful in the context of the software industry.

When the advantages described in the research and the theoretical description of open innovation are reviewed, one common theme clearly stands out: the characteristics of open innovation all serve to reduce *transaction costs* associated with the transfer of knowledge both into and out of firms. Transaction costs, as best described by Coase (1960), are the frictional costs related to negotiating an economic exchange. In Demsetz’s (1967) famous example, these costs took the form of negotiations concerning which hunter is allocated which beaver hunting ground and the related expenses for demarcating those grounds. This example involves a variety of methods to reduce or eliminate these

costs. The use of strong private property rules, for instance, largely eliminates such costs, at least after initial allocations are settled. These rules are a hallmark of the approach that, in the software industry example, may be broadly termed the *proprietary model*. This is a system under which property, including intellectual property, is owned exclusively by private individuals or entities, with corresponding legal structures (such as IPRs) in place to support such ownership. The work of Romer (1990, 2002) is useful here for understanding the anticipated nature and impact of IP on economic growth. In describing a technology-centric model (i.e., one in which economic growth is driven by *endogenous* technological change), Romer notes that technology is non-rivalrous and observes that growth is driven by the accumulation of knowledge. According to this view, IP helps to protect that accumulation and growth.

Under the FOSS model, which serves as an alternative to the proprietary model, a variety of mechanisms have been developed to address such transaction costs. These tools allow writers of software code to specify, at the time of code creation, how subsequent programmers may use their work. Doing so means the derivative work creator need not worry about paying licensing fees, negotiating distribution rights, or facing the threat of a lawsuit over IP rights. Somaya and Teece (2001) point specifically to transaction costs in their paper examining the integration of inventions into products. Furthermore, while not specifically addressing innovation, Posner (2004) points out that transaction costs, along with the copyright and antitrust law systems that they create, are leading to an inefficient distribution of IP among holders.

The Importance of Commons in Innovation

In the open paradigm, a possible solution to the challenges presented by many of these innovation issues (e.g., transaction costs and the treatment of IPR) presents itself in

the form of the use of knowledge as a commons. As described by Ostrom (1995) and others, the knowledge commons is a knowledge-based resource that is commonly accessible by a group of people and hence subject to governance and management by that group. Hess and Ostrom (2007) go further in describing the *meaning* of the commons: in their research, a commons is understood as a particular method of organizing economic activity around shared resources. They assert that self-management by the members of the commons community avoids the tragedy of the commons (Hess and Ostrom, 2007) – the idea first propagated by Garrett Hardin (1968) that suggests that in the pursuit of common resources, self-interest leads to the over-use of those resources. While Hardin might have argued for somewhat different measures (i.e., complete privatization or government regulation) to avoid the tragedy of the commons, Ostrom (1995) points to several historical examples in which common resources have been successfully managed by their care-takers and uses those examples to build an alternative model of governance. Such a system is superior to leaving knowledge management to elected officials or other regulatory forces.

A different commons-related challenge may explain why concern over the tragedy of the commons may not be sufficient to justify the use of IP. Heller (1998) refers to the ‘tragedy of the anti-commons’ in his study of the privatization (or lack thereof) of retail districts in post-Soviet Moscow. This research reveals that conflicts over ownership rights resulted in empty storefronts and commercial districts that would be derelict save for the presence of street vendors who could skirt the myriad rules and regulations. Heller and Eisenberg (1998) undertook a quantitative analysis of the anti-commons in biomedical research, finding that the increasing use of patents in that industry is crowding

out innovation. Murray and Stern (2007) encountered similar results in a test of how IPRs, in the form of patents, affect academic publication. Here the anti-commons effect was the result of investigators not being able to publish new research due to patent concerns. The tragedy of the anti-commons also describes the way in which IP laws can harm the flow of knowledge and information. In the software industry example, more recent research suggests a similar effect. Schweik (2011, 2013) offers the FOSS community as the standard for what a knowledge commons should look like. Schweik (2013) also provides an empirical study of the FOSS commons, demonstrating the way in which knowledge is shared by programmers across different software projects – a sharing that would not be possible in the presence of strong IPR protections.

While these arguments are persuasive in showing that IPRs can harm innovation (Heller and Eisenberg, 1998; Murry and Stern, 2007), they also demonstrate that FOSS allows knowledge to be shared better (Schweik, 2013). However, they do not offer evidence supporting any innovative superiority of the open innovation model used by the FOSS community or any other open innovation community. They also do not explain if or how the commons reduces transaction costs, thus enabling programmers to more easily take advantage of all of the external knowledge offered by the open innovation paradigm. In survey research of software programmers, Schweik and English (2007) suggest that the FOSS commons is more innovative, specifically citing the avoidance of transaction costs; however, this study only reflects the beliefs of the subjects surveyed.

The debate over whether the open approach is more innovative may therefore be framed as one of the tragedy of the commons vs. the tragedy of the anti-commons, with each possible innovation outcome combination being described by a corresponding

tragedy or lack thereof (see Table 1). In the classic viewpoint (or the status quo of current IPR policy in much of the world), the greatest concern is over Hardin's (1968) idea of the tragedy of the commons, as illustrated in the right-hand cell in the first row of Table 1. To avoid the tragedy, policy makers have traditionally chosen to move toward a proprietary knowledge approach, with the assumption that doing so would improve innovation. However, such a policy regime may in fact have two possible results. It is true that IPR rights may protect profits and encourage innovation, as shown in the left-hand cell of row two.

Table 1: Theoretical concepts: Tragedy of the commons and anti-commons

	More Innovation	Less Innovation
Knowledge Commons (such as FOSS)	Open innovation in the product leads to the introduction of significant external knowledge in the form of new innovations No tragedy of the commons (Ostrom)	Reduced incentives to innovate due to inability to secure profits through limited monopoly Tragedy of the commons occurs in products (Hardin)
Proprietary Knowledge	IPR protects profits, Encouraging Innovation No tragedy of the anti-commons (Status quo/Romer)	Knowledge flows are stifled by IPR Tragedy of the anti-commons (Heller)

However, knowledge flows could alternatively be stifled, thus harming innovation as in Heller's (1998) tragedy of the anti-commons (see the right-hand cell of row two). The commons approach can avoid these challenges by, as suggested by Ostrom (2005),

allowing flows of external knowledge to encourage innovation, which avoids both the tragedy of the commons and the tragedy of the anti-commons.

One of the main goals of innovation research is to provide evidence regarding where open innovation, as exemplified by the FOSS community, falls within the matrix presented in Table 1. Is it indeed Ostrom's scenario that prevents the tragedy of the commons from occurring, as is often suggested by proponents of those systems of innovation? Or are other factors more important, causing a different outcome? Advocates of the commons approach are more concerned with – and seek to avoid – the tragedy of the commons, while their proprietary model counterparts are conversely more concerned with the tragedy of the anti-commons. There is as of yet no empirical test of which claim prevails (or is more significant), which is a key gap in the existing literature. While that question is not directly addressed by this dissertation, a measure of external knowledge contributions nonetheless adds to the understanding of these questions.

The software industry is an ideal place to explore the above-mentioned issues for three reasons. First, the industry has a notoriously robust and widespread open source sub-sector – the FOSS community. Second, despite this open community's strength, it exists alongside an established and strong proprietary sub-sector. Finally, in an increasingly digital and technology-driven economy, an industry such as software is becoming more typical of the larger economy.

The Software Industry

According to the Business Software Alliance (BSA), the software industry had a total economic impact (direct and indirect) of more than \$1 trillion in the United States in 2014. The industry is also credited for creating some 2.5 million direct jobs and another 7.3 million indirect jobs and attributed an R&D spend of \$52 billion, which represents

more than 17% of total U.S. R&D spending in 2014 (BSA, 2015). The size and impact of the industry are also continuing to grow. As noted by the BSA's sister organization, the Software Industry Information Alliance (SIIA), "From 1997 to 2012, software industry production grew from \$149 billion to \$425 billion; and since this growth outpaced the rest of the economy, the software industry's direct share of U.S. GDP increased from 1.7 percent to 2.6 percent, or (an increase of) more than 50 percent"; moreover, 2.2% of all jobs in the United States were in the software industry in 2014, compared to only 0.9% in 1990 (SIIA, 2015).

These raw numbers tell a compelling story about the scope and impact of the software industry. However, a perhaps more interesting story emerges with a broader, more holistic understanding of what the 'the software industry' truly is. This term can in fact encompass a wide variety of firms, companies, and organization that are responsible for writing and publishing new software applications – regardless of whether their primary business goal is in fact to create and produce software. Firms dedicated only to this task are far from the 'norm' in the industry; firms ranging from insurance companies to healthcare companies to manufacturers develop software, often for their own purposes but sometimes for commercial sales. For example, according to the U.S. Bureau of Labor Statistics (BLS) in 2010, only 51.3% of software developers were employed by companies that could be identified as software publishers or computer designers (or the like). While this sector was indeed the largest employer of such experts, the figure demonstrates the diversity of players in the software market (BLS, 2013).

As software development is a relatively young industry, some innovation-focused concepts and policies were simply ported over to it from other fields at the dawn of the

computer age. Most relevant to this research is the fact that copyright laws are applied to software in largely the same fashion as in other industries. This means that software is subject to copyright protections for up to 95 or 120 years, depending on the situation. Considering that the shelf life of innovations in the computer and software industries is often measured in months and not years, this type of protection may seem inappropriate and excessive to some observers. Nonetheless, in the previously mentioned proprietary model of software development, most software has traditionally been developed with such copyright protection in mind. As noted earlier, the core defining characteristic of the proprietary model is the treatment of property in general and IPR in particular. The belief behind this approach is that without just compensation, programmers have little incentive to produce useful software and thus IPRs are necessary to secure this compensation via licensing fees or other similar methods. Microsoft founder Bill Gates, whose company is a notorious protector of its IP, staked this position out as early as 1976 in an open letter to computer users: “Is this fair? One thing you don’t do by stealing software is get back at MITS for some problem you may have...One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing?” (Gates, 1976). This belief corresponds well to the stated purpose of U.S. copyright (and more broad IP protection) law, which is to encourage innovation and new products by protecting creative and other works.

However, by the early 1980s, as companies sought to increase profits by taking advantage of the legal monopoly granted by copyright laws, some programmers began to see the situation as an affront to their freedom to work. The reasoning this time was that copyrighting software code prevented follow-on innovation (i.e., derivative works based

on earlier software code) and slowed the free dissemination of ideas. One such programmer was Richard Stallman, who would go on to found the Free Software Foundation (FSF) and with it introduce the Gnu Public License (GPL), which was one of the first FOSS licenses. Stallman created the GPL in an effort to free software code from proprietary controls and allow it to be used by future innovators. By doing so, he helped spawn the FOSS movement, which now includes independent programmers, large corporations, and the host of alternative licensing schemes that are collectively known as ‘copyleft’ (Williams and Stallman, 2010). Indeed, the FOSS movement initially concerned freedom, not creating a new business model. The formation and potential utility of the FOSS community for open innovation is discussed in greater detail later in this chapter. However, it is important to note that the creation of a new business model that presents a real competitive challenge to the proprietary model is exactly what has resulted from the FOSS movement. Even Microsoft, as renowned as it is for protecting IP, has allowed the FOSS model to creep into many of its products – which is a dramatic endorsement of this alternative model.

The FOSS segment thus appears poised for growth within the software industry. At the same time, the prominence and role of the industry itself continue to grow within the larger economy in both direct and indirect importance. This growth is part of the larger shift toward an information- or knowledge-based economy, a phenomenon that has been in progress for several years now. The Organisation for Economic Cooperation and Development (OECD) notes that there are three types of knowledge-based capital (KBC): computer information such as software and databases, IP such as patents and copyrights, and ‘competencies’ such as networking, brand equity, and other organizational know-

how (OECD, 2014). Herein may lie the most important contribution of this research, which deals at length with two of these three types of KBC. As the software industry is a major part of the information economy, how the treatment of knowledge affects innovation within this industry can describe and predict how innovation may affect other segments of the knowledge economy. A 2014 report from the OECD notes that “Innovation-based growth, underpinned by investments in a broad range of knowledge-based capital (KBC), is central to raising long-term living standards” (OECD, 2014). An exploration of open innovation in the software field can contribute to this important task.

The Free and Open Source Software Community: Powering Open Innovation?

A model of open innovation proposed by von Hippel (2007) refers to the ‘democratization’ of innovation – a process through which innovation by the user has become increasingly easy and commonplace. The user’s ability to innovate has been increased by a number of factors, many of which are the result of technological advancement. Easy access to both the tools necessary to innovate and the information or knowledge required to do so has been a major catalyst. This means that unnecessarily blocking users from innovating removes some of the most capable actors from the system. This is what the FOSS model hopes to avoid by allowing the user to enjoy the same abilities as the producer to utilize knowledge to build new innovations. But how will this process work in practice?

The FOSS model is viewed as being likely to encourage innovation in two major ways. First is the manner in which it requires information to be revealed freely. This necessary characteristic of a successful user-driven innovation system (which is advantageous for innovators in that it allows for new combinations of information and

reduces the probability that multiple innovators will work on the same task without collaborating) is directly at odds with proprietary IP protections. In cases such as FOSS, however, some manufacturers or producers choose to freely reveal and share the information (here the underlying code) that they used to create their product. Once this happens, users and others in the system can utilize the information for their own purposes – including to share their own (external) knowledge for free in exchange for the access granted. But why would an information owner be motivated to share freely? The answer is that the innovator actually benefits from sharing in a number of ways. Most notably, and most in conflict with the traditional model, FOSS supporters claim that free sharing leads to increased profits. Indeed, some empirical evidence (i.e., Dosi et al., 2006; Bessen and Hunt, 2007; Bessen and Maskin, 2009) supports this notion by showing that firms rarely use patenting and copyrighting to achieve profitability, due to the high costs of obtaining these protections. Evidence shows that the information protected by such measures tends not to remain ‘secret’ for long – generally no more than 18 months (von Hippel, 2005). Freely revealing information can also be beneficial to the innovator in less tangible ways. For instance, the innovator gains access to additional input from others that helps to strengthen his or her product. Software programmers often use input gained in this way to eliminate bugs in or make improvements to their programs. Furthermore, if the quality of a product is high, freely sharing is critical in building the innovation’s reputation and making his or her services more valuable and lucrative. Finally, integrating a product into the community, particularly if the innovator is the first to reveal a certain important piece of information, can lead to its widespread adoption. An example of this in the FOSS world is the Linux kernel, which was considered a breakthrough

innovation when it was introduced. Despite giving their source code away for free, the developers of Linux have still parlayed their success into a billion-dollar enterprise (mostly by selling service and support).

The benefits of free sharing are supported by the empirical work of Henkel (2006), who found in a study of 268 firms using the open source Linux operating system that more than half of those firms freely shared their developments and customizations of the tool. According to Henkel, a firm's decision to share depended on several factors, including the intended use of the code that it had developed and the firm's characteristics. Most interestingly, the decision as to whether to reveal information changed in different situations within the same firm; most of the firms surveyed successfully shared some information while protecting the rest proprietarily. This suggests that firms' decision makers do not opt to share freely based only on their own perceptions of it being the 'right thing to do'; they are instead making rational business decisions based on identified benefits. Responses to the survey support this conclusion. While several respondents did note that they often share their code because many FOSS licenses require them to do so, the other top responses included "revealing good code improves our company's technical reputation" and "visibility on the mailing list is good marketing" (Henkel, 2006). These responses closely mirror the benefits of free sharing put forth by von Hippel (2005).

The second way that the FOSS model encourages innovation is through the creation of *innovation communities*. As envisioned by von Hippel's (2005) model, an innovation community is a group of users who share an interest in a given product or field but have diverse backgrounds that complement each other and allow them to offer different inputs that help to create a better overall outcome. Community members may or

may not freely reveal information to each other, although it is necessary that at least some do so. The FOSS community enjoys a strong tradition of sharing, which may be one reason that this community is so strong. In describing how this community helps foster innovations in software, von Hippel notes that new programs often result from a programmer observing a need for a piece of code that does not yet exist. This should not be considered a particularly exciting observation; many innovations begin with an innovator catching an un-filled need or demand, regardless of the industry. However, in the case of software, particularly FOSS, the need is often felt by multiple users simultaneously. Due to the strength of the community, these users can easily locate each other and begin collaborating to find a solution. The early collaborators often become the ‘gatekeepers’ of the project, managing and coordinating efforts and communication (von Hippel, 2005).

Indeed, ample evidence demonstrates how collaborators or users with similar needs can group together and become powerful innovation forces even within an open innovation community. Baldwin and von Hippel (2011) discuss how the phenomenon of modularity helps to transition an industry from producer-centric innovation to user-centric innovation. They argue that if a system is modular, the many tasks involved in production are allocated to different groups – and it is likely that the group with the proper skills and knowledge to tackle a particular aspect of any given problem will become the module that addresses that challenge. In this way, modularity both encourages collaboration and eliminates inefficiencies in the system. Baldwin and von Hippel (2011) find that the modular orientation of an open innovation system is one of several factors that drive production costs down and make this model superior to

producer-based innovation. When open innovation is present, it is likely that many producers will either exit the market or change their business model (possibly by joining the community) to stay competitive.

Another element of the community that is of interest to innovative potential is where those community members come from, that is, their affiliations with firms, organizations, and other entities beyond the group that is responsible for managing the open innovation project in question. For example, Von Krogh et al. (2003) note that affiliation is important to specialization in FOSS products. Their study only explored community members' affiliations, not their knowledge contributions. Combining affiliation and contribution could lead to interesting conclusions regarding communities, openness, and innovation.

Returning to the core issue of innovation in the FOSS community, recent empirical evidence supports the conclusion that the community can be leveraged to the benefit of an individual firm. Dahlander and Wallin (2006) utilize the innovation framework of Teece (1986), which emphasized appropriability and complimentary assets as the keys to a firm's ability to profit from innovations. Dahlander and Wallin (2006) seek to understand how open innovation communities such as FOSS leverage their users as complimentary assets. To achieve this, the researchers closely tracked and monitored a substantial group of FOSS users and programmers who were working on a common project, namely the GNOME desktop environment. Their study entailed cataloging which programmers were sponsored by firms and which were not and subsequently analyzing their behavior within the community. The results demonstrated that sponsored individuals are significantly more likely to have higher numbers of contacts and interactions within

the community. Furthermore, sponsored individuals are also significantly more likely to be contacted by other community members. It seems that firms are indeed able to leverage the activities of their employees within this community in order to benefit their own business.

In order to further understand an individual's exact position in the user community (e.g., at the core, on the periphery, or connected to a firm), Dahlander and Frederikson (2012) conducted research that focused on the employees of a Swedish software firm that was immersed in the FOSS open innovation community. The goal of this study was to determine how an individual's propensity to innovate is affected by the level and nature of his or her activity in the community. The researchers hypothesized that either being located within the user community's core group or being a member of several external related communities would enhance innovative activity, at least to a point. They further speculated that membership in one of these groups would be mutually exclusive; individuals attempting to become core members of the community would not be able to be more innovative by also joining peripheral communities. To explore these questions, the study tracked the interactions of the firm's employees, who were also asked about their activities in the community. The results demonstrated support for both hypotheses. Core members and members connected to several peripheral groups were all more innovative than their peers. However, a substitution effect was also observed, as core members were unable to derive additional benefits by becoming active in peripheral communities and vice versa (Dahlander and Frederikson, 2012).

Adding to the literature that demonstrates the innovative power of open communities, Boudreau (2012) undertook an extensive quantitative study of the

producers of software “apps,” the now ubiquitous programs written specifically for mobile and other computing applications. The explosive growth of these programs has largely mirrored the rise of the FOSS movement, which is no surprise given that many mobile operating systems (OS) – including the most popular, Google’s Android OS – are open source. The open source orientation of the app writing community may have contributed some selection bias to this study, but it also presented a unique opportunity to study a community with few pre-existing notions of how business and innovation work. Boudreau (2012) was able to obtain data on the publication and sales of mobile apps over a period of six years. This was compared to a number of innovation-causing variables, such as the number of producers on a given platform, the time it takes to produce new versions of software, and the scope (i.e., number of genres) of the different applications being worked on by a given producer.

The results of Boudreau’s study demonstrate the importance of the open innovation system. Perhaps most tellingly, the amount and variety of software being produced within a given platform/OS is directly linked to the size and diversity of the communities of programmers working on that platform/OS. This speaks to the innovative power of the open source model: adding more human capital makes a system more dynamic and innovative. Boudreau (2012) found that two attributes of the open innovation system were critical for producing this outcome, namely lower barriers to entry and ease of access to information. Open innovation communities achieve these things by reducing both the knowledge needed to participate or innovate (because knowledge is shared) and the required capital resources (as resources can be pooled). The result is that more potential innovators are able to join and contribute their external

knowledge to a community. As this study focused only on the production of new apps, it omitted a key piece of the equation: how increasing creative output could be converted into a profitable business model. Lamoreaux and Sokoloff (1996) touched on this topic when they noted that there is a growing divide between those who create innovations and those who are able to exploit them commercially. The open innovation model may portend a deepening of this divide. Regardless of how profits may be accrued, the effect on innovation is strong enough to suggest that open innovation networks represent a very real potential business model.

A community may also be able to enhance its output beyond that of the individual like-minded persons who constitute it. By using the complementary skills of its members, a community in a way becomes a whole that is greater than the sums of its part. Research shows that the FOSS world features ‘communities of interest’ (COI) (i.e., McDaniel et al., 2006; Amin and Roberts, 2008) that achieve just this sort of enhancement. Less clear, however, are the mechanisms behind why these COIs form and how they may affect innovation. This dissertation explores those questions, specifically seeking to fill the gap in the literature concerning new ways in which FOSS orientation can enable paths to innovation that are not open to proprietary development systems.

The curious observer may note that an open innovation community would seem to suggest serious competition between rival producers; after all, competitors could easily obtain and use source code in their own product. Such competition can have a mixed effect on – or even be harmful to – innovation. For example, Aghion et al. (2005) suggest that an inverted-U relationship exists whereby some competition helps innovation, but too much can wipe out any gains by discouraging lagging firms and creating industry-

wide equilibriums that are out of balance. According to Boudreau (2012), this is not the case in an open innovation environment, where producers share information and work together through other means so that more producers means a more powerful and innovative network. Von Hippel's (2005, 2007) finding on competition in the innovation community mirrors this analysis. Even when firms engage in rivalrous battles over market share, they tend to share information and collaborate in open innovation settings.

Open innovation, knowledge commons, and open source are the three ideas that provide the foundation for this dissertation. These concepts are interesting when taken separately; understood together, however, they point to critical issues concerning how knowledge can be treated to maximize innovation and pose the important questions that are explored in this dissertation. These questions are answered using the above-described lens to undertake an in-depth case study of a FOSS product/community.

CHAPTER THREE: UNDERSTANDING INNOVATION IN FREE AND OPEN SOURCE SYSTEMS: A CASE STUDY METHODOLOGY

Introduction

The case study in this dissertation uses both quantitative and qualitative measures to understand open innovation in the FOSS environment. This chapter describes in detail the empirical approach utilized to collect and analyze quantitative and qualitative data, (including how they are blended together in the analysis). A brief discussion of how related studies of innovation have measured this concept first offers a glimpse into the challenges inherent in that task, which helps to set the stage for understanding the value offered by the novelty of the approach used in this dissertation. Reviews of the research question, hypotheses, case selection, and variables subsequently provide further insight into the value and uniqueness of the research. The chapter concludes with a discussion of the actual structure of the research, including the validity checks undertaken.

The Challenge of Measuring Innovation

Measuring innovation is an inherently challenging task due to the difficulty in defining the term ‘innovation.’ Is any new product an innovation? Does that product have to be revolutionary, popular, or profitable? Is a new medical device innovative if it saves lives? Is the newest Apple product innovative? No single good answer to these questions exists, but innovation must still be measured – and some sound approach must be used to do so. A commonly accepted measure utilized by researchers studying innovation is a firm or industry’s R&D spending. The idea here is that as more energy and resources are invested into producing new goods and ideas, the chances of innovation occurring improve. Many researchers have used R&D spending to study innovation both quantitatively (e.g., Sampson, 2007, whose study measured the impact of R&D spending

alliances between firms in the telecommunications industry) and qualitatively (e.g., Kriaa and Karray, 2010, whose case study compared innovative effects in two firms with different approaches to R&D). Indeed, R&D has historically been a popular choice for measuring innovation among researchers at the U.S. National Bureau of Economic Research (NBER). Mansfield's (1984) research emphasized the importance of basic research; Crepon et al. (1998) found significant firm-level effects on R&D spending and subsequent innovation; and Hall and Lerner (2009) explored gaps in the market for the funding of and investment in innovative activity. Jaffe (1989) used the R&D metric to investigate the impact of academic research spillovers on industrial innovation, finding significant positive effects; nonetheless, he would later warn in 2011 that R&D spending can be a problematic measure of innovation, in that "subtle and difficult issues that must be attended to using appropriate statistical techniques as well as by using multiple and diverse quantitative metrics" (Jaffe, 2011). Wallsten (2000) was another to use this measure, in his study of the U.S. federal government's Small Business Innovation Research program.

While R&D inputs remain a popular gauge of innovative capacity, or at least innovative potential, for the purpose of this dissertation this measure fails for one clear reason: open source firms *intentionally* eschew traditional forms of R&D activity, particularly with respect to the use of external knowledge (which is a key variable for this research). Spending on such ventures therefore does not serve as a reliable indicator, which means another measure must be found. Fortunately, as with most any issue, more than one way exists for measuring the level of innovative activity that is being undertaken by a firm, industry, or economy. For instance, instead of measuring innovative inputs

with R&D spending, it is possible to measure innovative output. However, defining what exactly constitutes an innovation still remains a challenge. One suggestion is that patents and copyrights constitute a good measure of innovation. Advocates of this concept argue that a new patent or copyright means a new idea or product. This metric equates, for example, one patent to one innovation, regardless of the relative merits of the product or idea patented. According to this definition, a device that may be thoroughly useless to society is still considered an innovation so long as it achieves patentability. But is this an accurate measure? Bessen and Hunt (2007) used patenting as a proxy for innovative activity in the software industry. In their study of firms, they actually found that patents are a poor measure of such activity. Patents are generally only used by a small group of firms whose business model is focused on strong property rights protections. In their sample, Bessen and Hunt (2007) found that only about 5% of firms patenting software were software publishing firms; the rest were manufacturing firms of various types.

The persistent challenge of using patents and copyrights as a measure of innovation likely stems directly from the nature of these legal devices. On the one hand, the very purpose of patent and copyright systems is to protect IP, contingent to the theory that IP rights are necessary to secure profits and therefore encourage innovation – which makes them a logical choice for measuring innovation. On the other hand, patents and copyrights do not signal innovation per se; they merely offer legal protection for a creator's work. As such, they are only as useful as the idea or product that they protect. If most patents are awarded to highly innovative products and most copyrights go to creative new works, then they are an excellent measure. Of course, the opposite holds true as well.

Nevertheless, many scholars continue to use these measures to gauge innovative activity. Mossoff (2013) touches on the underlying debate over the utility of IP protections for producing innovation, focusing on the case of copyrights and academic research. As an economist, Mossoff stresses that incentive-to-create is not the only reason to have such protections, noting several policy implications of doing away with copyright. Even if creators are motivated to carry out their work without a profit-securing copyright, the infrastructure costs involved with production and dissemination would demand some kind of financial support. If we are to believe this argument, then even when the creator of an innovation does not seek to exploit his or her work for financial gain (as is the case for many software publishers), a copyright may still be necessary to support the industry and infrastructure that transform invention into innovation.

In reality, the utility of this measure depends on how inventors use patents and copyrights. Teece (1986) shows that IP protections are only one possible way to secure profits from innovation in technical fields. The overall business plan and appropriability regime employed by a firm can utilize a number of strategies, including strong appropriability through IP protections. Cohen et al. (2000) discuss these issues in their study of why U.S. manufacturing firms do or do not use patents. In addition to discovering that patents are the least likely method to be used by these firms (behind inter alia complementary marketing and processes, trade secrets, and lead time), the research demonstrated that significant differences in practices exist in assorted industries. Further differences were evident when firm size was taken into account.

Given the challenges that exist in conjunction with describing the innovation relationship in a purely quantitative fashion, using a more qualitative or mixed approach

was considered for this dissertation. Such an approach offers advantages over a purely quantitative study; the primary benefit is that it enables data and consistency issues to be avoided, although it is equally important that it has a proven track record in answering questions related to innovation and open source communities. For example, Sundstrom and Zika-Viktorsson (2009) used a case study to demonstrate how FOSS projects organize to increase innovation. Piva et al. (2012) utilized multiple case studies to demonstrate relationships between FOSS communities and innovative entrepreneurs. Other examples of FOSS innovation research using case studies include Dahlander and Magnusson (2005, 2008) and Edison et al. (2013).

Of the many studies that have used qualitative and case study approaches to answer questions of open source and innovation, the research methodology employed by von Krogh et al. (2003) most closely resembles the approach used in this dissertation. In that previous research, von Krogh et al. (2003) used a case study methodology to explore why individuals or groups join existing communities of software developers as well as the ways in which those subjects then participate in terms of contributing code to projects. While the study methodology and data collection approach are in fact largely similar to those employed in this dissertation, the thesis presented and questions posed by von Krough et al. (2003) diverge significantly; in particular, that study did not answer questions related to improving innovation outcomes or how FOSS encourages such activity.

The variables measured in the current research and the study of von Krough et al. are similar and in many cases the same. They include code contributions, the numbers and backgrounds of contributors, and information about the types of contributions. Von

Krough et al. (2003) measured these and other items by selecting an open source software project (in this case Freenet, an online communications and chat tool) and harvesting the archives of email communications between members of the open source community. They then coded the information within those emails to produce a vast database that describes the community's interactions. This database was augmented with information from interviews with a variety of community members at both the core and the periphery of the project. As noted above, von Krough et al. (2003) did not measure innovation as part of their study. However, the methodology they used can be adapted for that purpose in a similar study, such as this dissertation.

Research Question

The research of Ostrom (1990, 1995) best identifies the unique compilation of individuals and knowledge known as the commons. Simultaneously, the research of Chesborough (2003) explains the phenomenon of open innovation and suggests that open systems produce greater innovation. In order to understand important policy issues, the primary question that this dissertation seeks to answer is as follows:

R1: Does participation in a knowledge commons in the form of the free and open source software community affect innovative potential?

Simply put, the task of this dissertation is to demonstrate how the commons *might* enable open innovation systems to create greater innovation outcomes. The main line of inquiry into this relationship involves understanding how the introduction of external knowledge from the commons adds to the innovation system. As such, the following related question is also considered:

R2: How does external knowledge from the commons community affect innovation potential in software?

In answering these questions, this dissertation provides insight into a host of issues, including open source software development, innovation, and IPR management.

Hypotheses

To determine whether some inherent aspect of the FOSS community's nature may be responsible for an increased innovative potential resulting from external knowledge etc., it is important to explore how open innovation works in general as well as the open innovation/knowledge commons found specifically in the FOSS community. The FOSS community is a somewhat more complicated entity than the simplification presented by Stallman's 'free as in freedom, not free as in free beer' analogy and thus requires a more nuanced description. In his seminal work *The Cathedral and the Bazaar*, Eric Raymond (1998) described the FOSS community as the 'bazaar' compared to the 'cathedral' model of what he categorizes as the commercial software industry. The analogy of the bazaar was arrived at through his experience working in the Linux community, which was then – and likely still is – the largest FOSS project in the world. Raymond intended the cathedral to represent the traditional method of building software, in that software is constructed painstakingly over time by a small band of specialists and is not available for use until the last brick is laid (much like a medieval cathedral would have been built); in contrast, the bazaar was meant to describe a community in which people with different backgrounds, skill sets, and agendas are able to work together in a coherent (if sometimes confused) fashion to produce an outcome that is useful to all – and in fact superior to that of its rivals. Raymond's work is perhaps the earliest attempt to describe how the FOSS model successfully competes with the proprietary alternative. However, while it is foundational and essential for understanding the workings of the Linux and FOSS communities, Raymond's research is not an empirical study that offers proof in support of its claims.

This bazaar analogy is nonetheless a natural starting point for developing a further understanding of the FOSS community. The FOSS community is indeed a diverse set of people who are sometimes working toward opposite goals but end up helping each other along the way. However, this understanding is incomplete and incorrect in that it misses a critical component of economic growth. That is, while Raymond describes an environment in which innovation thrives and new products are created efficiently in spite of the bazaar's seeming chaos, he does not mention how it can be successfully converted into a working business model that is sustainable in the long term. True, he observes that programmers who love the projects they work on are more engaged than those who only do jobs they are paid to do. Nonetheless, it is difficult for the former programmers to write code if they are unemployed and starving.

While this dissertation is not about how to profit from the FOSS model, the profit motive is a strong force and innovation is tied into profits in many ways. Clearly it is possible to profit using a FOSS model. RedHat Inc., the makers of Linux, recently surpassed \$1 billion in annual revenues (McMillan, 2012), based almost entirely on the support services they provide. This business model seemingly assumes that if enough people want to use a firm's product, at least some will be willing to pay for technical support from the experts. Assuming that building a strong user-base requires an innovative product, it is easy to see why innovation retains its importance. The critical question thus remains: What about the FOSS community can encourage greater innovation?

Such a question can be asked on multiple levels. The most important of these are the product and firm levels, which this research addresses in turn. At the product level, a

product innovation can be defined in a number of ways – and much like innovation in general, entire scholarly works and books are dedicated to doing so. However, a good starting point is the definition used in the OECD’s *Oslo Manual: Guidelines for Collecting and Interpreting Innovation Data*, which describes a product innovation as: “A good or service that is new or significantly improved. This includes significant improvements in technical specifications, components and materials, software in the product, user friendliness or other functional characteristics” (OECD, 2014). The current study is then an attempt to describe how the use of external knowledge by the FOSS system increases the number of these ‘significant improvements’ being made to software programs. An alternative way to approach the question is to describe an increase in the overall innovation potential of the entire group working on a particular project. The first two hypotheses are therefore:

H1 – Software programs produced in the FOSS commons feature more innovation than non-FOSS software due to direct contributions of external knowledge from the commons

H2 – Software programs produced in the FOSS commons feature more innovation than non-FOSS software as the addition of external knowledge from the commons better enables internal innovation

To describe H1, it is necessary to understand how external knowledge affects innovations originating directly in the FOSS community (i.e., outside of the firm or core development group). In H2, the task is comparatively more difficult, as the goal is to describe how the FOSS community contributes more abstractly to the body of knowledge and results in a firm or core group that can itself innovate more easily or in new and

different ways. A finding that confirms either H1 or H2 – or more accurately, that fails to reject the associated null hypotheses – would suggest that the open model successfully encourages innovation. In this case, the benefits of the commons would outweigh the loss of profits associated with more stringent protection of IPR and the knowledge held in a commons as described by Ostrom (1995) could be seen as a cause of greater innovation.

Of course, these tests are designed to assess the relationship between the use of knowledge commons (as measured by the contributions of FOSS external knowledge) and innovation at the product level. Simply put, confirmation of one of these hypotheses will reveal whether products developed as FOSS are likely to be more innovative than non-FOSS products. In examining the difference between product innovation and firm innovation, it is obvious that things are more complicated for firms. Innovation may or may not be a specific business goal for firms. Furthermore, whether to use commons knowledge is a decision that each firm makes independently, based on its specific needs. Some firms, such as RedHat Inc., have built their entire business model on FOSS. As previously noted, others (e.g., Microsoft) are notorious for protecting their IPR. However, the vast majority of software firms in the industry today fall somewhere in between: they use some commons knowledge (i.e., FOSS) in certain parts of their business while still applying IPR in other parts. In seeking to understand how knowledge commons are used in open innovation systems, it is therefore critical to measure how firms' treatment of FOSS affects innovative output.

By examining how IPRs are addressed by the firm that produces the software program, rather than in the product itself, this research describes whether knowledge commons can be integrated into a business model that successfully promotes innovation.

While the same basic research question is still under examination, the hypotheses are now as follows:

H3 – A firm's participation in the FOSS commons allows it to absorb more external knowledge that can be used to produce more innovations

H4 – A firm's participation in the FOSS commons produces outcomes that allow it to better allocate existing resources in producing new innovations

The results here demonstrate an effect that is independent of the effects measured by H1 and H2. In particular, they demonstrate to what extent, if any, a difference between product and firm innovation exists in the software industry, providing a chance to evaluate the level at which the impact of external knowledge from the FOSS community affects innovation.

The difference between product and firm innovation is of crucial importance in understanding how the FOSS commons may be driving innovation. If firm effects are playing a more important role, adopting and governing a business model would appear to be the keys to harnessing the power of knowledge commons. A finding that the general impact on the product is more important, however, suggests that the model and its governance are less important, and that simply being a part of the commons is the deciding factor. In essence, this determination defines whether the knowledge input (i.e., the choice to use FOSS as a means of acquiring external knowledge) or knowledge management (i.e., how a firm uses the FOSS commons/external knowledge) is more significant. Table 2 below demonstrates how this dissertation measures both product- and firm-level effects.

Table 2: Description of empirical measurement

	More Innovation	Less Innovation
Software Produced in FOSS Commons H1 and H2	Open Innovation in the product leads to the introduction of significant external knowledge, resulting in new innovations Hypotheses supported	Reduced incentives due to inability to secure profits through limited monopoly results in reluctance to share knowledge; innovation suffers
Firm Uses FOSS Commons Prominently in Business Model H3 and H4	Open innovation by the firm allows it to use external knowledge to achieve superior innovation results Hypotheses supported	Too many firms attempt to share IP and none can profit, which crowds out innovation despite the presence of external knowledge

The effect of the presence of external knowledge is described in each case included in the table. In column one, software is produced in the commons with the presence of external knowledge; the corresponding hypothesis of this dissertation would be confirmed if greater innovation outcomes result. The converse of this is shown in column two, which describes possible negative outcomes of the open approach. If lower innovation results, IPRs appear to be needed to protect profits and therefore innovation. The corollary is that innovation would be difficult in the open regime, where no such protections exist. Findings that innovation thrives in the FOSS world would call into question this idea about IPRs.

One claim regarding the connection between the need for IP and innovation stands out, namely that profits must be protected in order to encourage research and innovation. In discussing the problem of constructing a profitable business model from

FOSS, it should be clear that the profit motive is not irrelevant – it is simply not linked to sales of end-product software for these firms. For example, some research suggests that limited open revealing can actually increase profitable sales (Alexy et al., 2009). Furthermore, as noted above, firms such as RedHat Inc. have found other ways to profit. The profit motive thus remains, while transaction costs are driven down and innovation is maximized. This research then shows how the open/commons approach as embodied by FOSS helps aid innovation. It does not directly address the profit issue or explore whether the ability to profit without the benefit of IPRs may be the underlying cause of any IPR innovation uncoupling (as described below); these are tasks best left for additional research. Nonetheless, it is important to note that ample evidence shows that the FOSS approach does not prohibit profitability.

The FOSS community provides an outstanding opportunity to further the existing body of knowledge in relation to significant questions concerning open innovation and external knowledge. This research shows that the particular qualities and attributes of the FOSS community enable innovation, specifically by unleashing knowledge flows that are not possible otherwise. These knowledge flows include unique communities of like-minded individuals who both willingly share their external knowledge and work together to specialize in certain ways that enhance the FOSS project in question. The research also describes a model for understanding the governance of this open innovation mechanism in the FOSS community, and in doing so further contributes to the overall body of literature on open innovation.

Case Selection

In his work *Case Study Research: Design and Methods* (2013), Robert Yin describes the case selection process, noting that “the complete research design should

indicate what data are to be collected – as indicated by a study’s questions, its propositions, and its units of analysis. The design should tell you what is to be done after the data has been collected – as indicated by the logic linking the data to the propositions and the criteria for interpreting the findings” (Yin, 2013). Other issues that are critical to consider in case selection include unit of analysis, the study questions, validity (with external validity being of greatest concern), and access to data.

Von Krogh et al. (2003), used a deliberate approach to case selection. To begin, they selected a single case “in order to increase the depth of the analysis, acquire and report experience with the gathering of new and unfamiliar data” (Von Krogh et al., 2003). They specifically considered three criteria to be most important for their case selection. First, the Freenet project studied was a unique effort, which researchers believed would lead to novel results that addressed their questions. Second, the nature of the Freenet project suggested to the researchers that it was a suitable avenue for providing appropriate access to data and a clear unit of analysis. Finally, the type of activity and time period studied were similar to several other open source projects, which led the researchers to conclude that this was a case with good external and internal validity. This illustration makes it easy to see how case selection was first affected by research design, as well as how it also later affected the outcome of the research of von Krogh et al. (2003).

The implications from both the Yin (2013) and von Krogh et al. (2003) examples are clear. With these lessons in mind, this dissertation focuses on a single case study of the Snort FOSS program/project. Snort is an intrusion detection system (IDS) – that is, a program that detects malicious activity on a single computer or a computer network – that

was developed from its inception in the open source community. It can be used by individuals or firms to protect computer systems and network against potentially malicious activity (e.g., cyber attacks and hackers). While the program started as the work of a single individual and grew to be a flagship product of Cisco, one of the world's largest software companies, it has remained open source. The process that resulted in the Snort program being selected as the case to be studied in this dissertation focused on the following criteria:

- **Appropriateness for answering the main study question:** To answer the questions posed, the study subject should be a clear case of FOSS development and offer insight into how innovative development occurs in the project. Snort's administrators at Sourcefire, Inc. (part of Cisco, Inc.; hereinafter: Sourcefire) are enthusiastic supporters of FOSS development. At the same time, Snort is a highly successful project, even while competing with non-FOSS alternatives. The opportunities for contrast are significant, and the results are likely to address the research question as posed.
- **Access to data:** By their nature, FOSS projects tend to make information about their development more publicly available, and Snort is no exception. The primary means of communication for Snort developers is an email list, and every message sent using that list since its inception is archived in an online database. Furthermore, the main contributors to the project are active members of the open source community and likely willing interview candidates.

- **External applicability/validity:** Case selection is perhaps the most important part of the study design when it comes to external validity. While various methods for ensuring validity are available, as noted by Yin (2013) and others, it is most critical that the case selected be typical of the larger population that is being described. While FOSS represents a vast array of different projects and no one project can truly represent all others, the Snort project is a typical FOSS project in many ways. For instance, it uses the GPL v 2.0, which is the most popular FOSS license. Furthermore, it has been in existence since at least 2000, which is longer than most FOSS projects but not as long as many of the most common (e.g., Linux, SendMail, and MySQL, all of which have been around since the beginning of the 1990s or earlier). As a result, it can be expected that the findings of the Snort case can reasonably describe useful information about other FOSS projects.

Variables

One important commonality between this dissertation and the von Krogh et al. (2003) study is that both answer questions of *how* or *why*. In the case of the prior work, the question was “Why do subjects join an open source community?” In this dissertation, the question is “How (or why) might open source result in greater innovation outcomes?” Yin (2013) describes the advantages of the case study when answering questions of ‘how’ or ‘why’ as such: “In contrast, ‘how’ and ‘why’ questions are more *explanatory* and likely to lead to the use of case studies, histories, and experiments as the preferred research methods. This is because such questions deal with operation links needing to be traced over time, rather than mere frequencies or incidence” (Yin, 2013). In practice, Audretsch (1995) comments on the utility – even necessity – of utilizing longitudinal

databases to track firms or industries over time. As such, the use of a case study and an archival database are again ideal for being able to measure the behavior of an innovation system over a period of time.

The main independent variable in this study, namely external knowledge, seeks to describe how much innovation-inducing input (i.e., how much knowledge) has been drawn from the FOSS commons. In other words, what were the community contributions of external knowledge, above the internal knowledge developed within the core development team? The addition of knowledge that is ‘external’ may reveal itself in different ways, such as new lines of code that are written by a community member and ultimately added to the project or the fixing of security holes. Finally, in this spirit of Feldman and Audretsch (1999), Audretsch (1995) and others whose work has taken advantage of the U.S. Small Business Administration’s Innovation Database – a direct measure of innovation, a count of when and how many new features were added to the software program can describe new knowledge.

Of the three measures of innovative input described above, the first, new lines of source code contributed, is the most straightforward and easiest to define and measure. While not all code is alike or innovative, contributing new code to a program is the only possible path for adding a new innovative feature. While lines of code are thus not a definitive measure of innovation, they are indeed a sure measure of innovative input or potential – without new lines of code, there can be no new features. It is therefore reasonable to assume that a contribution of more lines of code has greater potential for additive innovation. Indeed, lines of code have been similarly used to measure the growth

and improvement of software programs in the past, notably by Koch and Schneider (2000), Dahlander et al. (2008), and Von Krogh et al. (2003).

The second measure, security bugs, marks an interesting and unique type of knowledge input. The specifics of this uniqueness are discussed further in Chapter Five, which focuses specifically on this type of knowledge. In this context, however, bugs are issues or problems with a program in its current form. The identification of such issues adds new knowledge by increasing the awareness of problems or weakness in the code. One method that aids in the identification of security bugs (although it is not definitive or exclusive to the task) is to search for key words and terms that the developers themselves use to highlight bugs they have found in the program, such as broke, bug, fix(ed)(es), killed, incorrect, correct(s), crash, troubleshoot(ing), typo, mistake, eliminate, and address. In the context of Snort, the coding process for bugs involved the coder searching through the messages for these types of signals. One important rule was included in the coding process to filter out false positives: any message from the archive that yielded a possible security bug submission was only included if further discussion in the community revealed a second independent confirmation that the issue did in fact exist (i.e., another user confirmed finding the same failure in the code or the Sourcefire team confirmed that they would add the bug to their list of issues to fix).

Finally, the third measure of knowledge contribution is the new feature that is developed. Features are new ideas or suggestions for enhancements to the existing program; they may include code related to their implementation but do not necessarily have to. In the Snort message archive, these features are easily identifiable in conversations among developers because those offering new ideas are eager to highlight

them for incorporation into the program. Similar to security bugs, certain terms and keywords often signal the presence of a new feature, such as new, allow(s), enable, feature, access, option, change, plugin, update, enhance(ment), and add(s). The distinctness of these terms from those that signal security bugs is important in the task of differentiating between new features and bugs. The difference can be further understood by noting that a bug is a problem that is getting fixed, while a feature is some useful new addition that is not necessary for a program to function well but does increase its utility in some way. In plainer terms, if an issue breaks a program or otherwise renders it useless, the corresponding corrective input is a bug. If it simply better the user experience, even in a small way, the input is a new feature. Similar to bugs, the coding of features employ the rule of dual-confirmation of the newness of the feature submitted to the community.

As noted previously, innovation, the dependent variable for this study, is difficult to measure in a quantitative fashion. Nonetheless, it can be understood in a number of ways for description. A broad, general understanding of innovation recognizes that increased innovation results from a number of different inputs. The basic relationship between innovation and the inputs described above can be expressed, in general form, as:

$$\Delta_{innov} = input_1 + input_2 + input_3 + \dots$$

This dissertation assumes that in the software industry, a FOSS development team has access to the same types of inputs as a proprietary development team. Both can hire software new developers, license IP, access capital, and collaborate with other groups or

firms. The differentiating factor is external knowledge from the FOSS community. Here the proprietary innovation relationship may be expressed as:

$$\Delta innov_p = internal\ knowledge + licensing + hiring + capital + collaboration$$

Where $\Delta innov_p$ is any change in innovation that arises from the proprietary development team. The FOSS innovation relationship adds another factor:

$$\Delta innov_F = internal\ knowledge + licensing + hiring + capital + collaboration \\ + FOSS\ external\ knowledge$$

Where $\Delta innov_F$ is any change in innovation that arises from the FOSS development team. This can be further broken down to demonstrate the effect of each of the three types of external knowledge described above, but such differentiation is not required to understand the innovation-external knowledge relationship. It is sufficient to recognize that all are measures of external knowledge that is only available due to the existence of the FOSS community.

Because the data collected for this study provides a usable measure of external knowledge and all of the other inputs are understood to be equally available under both the proprietary and FOSS models, this research provides an opportunity to understand the innovation outcomes in the FOSS system by simply quantifying and describing the nature of external knowledge. In this way, external knowledge inputs solve the innovation data challenge by serving as a viable proxy for innovation. By accepting that additional

knowledge enhances innovation (as demonstrated inter alia by Chesborough, 2003, 2006), the main task of this dissertation is then to demonstrate how and to what extent the FOSS system encourages the contribution of external knowledge.

As a result, it is important to ensure that the three measures of innovative inputs used in this study are external – not internal – knowledge. Here the goal is to measure the affiliation of the information's *contributor* in a binary fashion: the contributor is either a member of the FOSS community (with no direct affiliation to the project or the firm running it) or a direct affiliate of the project. The logical outgrowth of this is that inputs from outside contributors represent external knowledge, while inputs from internal contributors represent internal knowledge. Because the case selected was operated by one firm (namely Sourcefire) during the entire research period, in this study an affiliate is defined as a current employee of that firm. The term employee is generally used to denote such a person, but 'affiliate' should be understood to mean the same thing when utilized instead. Anyone who is not a direct employee of the firm is therefore connected to the project only via his or her participation in the FOSS community and thus considered a *non-affiliate* or *community member*. For the purposes of this dissertation, the work and the contributions of Sourcefire employees are considered internal knowledge while contributions from non-affiliates are considered external knowledge.

This gauge of affiliation may appear difficult to measure. The nature of the software industry in general and the FOSS community in particular presents some barriers, as individual contributors can move easily in and out of the FOSS community for any given project. Furthermore, contributions can even sometimes be difficult to quantify within the community. However, the nature of how FOSS communications are

preserved presents an opportunity to track the affiliations of contributors and thereby understand the role of the FOSS community. In general, FOSS development occurs in a collaborative environment; it is frequently coordinated through online communications, with communications being archived in databases for future reference and historical purposes in case previous development discussions need to be consulted. The records normally display information such as user email addresses, and it is possible to use these identifiers to classify affiliations. It is an archive such as this that von Krough et al. (2003) used in their research to effectively describe a host of variables.

Study Design

The Snort project, which dates to founder Martin Roesch's work in the late 1990s, became a fully FOSS project in 2000. Since that time, the central mode of communication for software developers seeking to participate in the project has been an email list, also known as a listserv, called *Snort-devel* (email address: Snort-devel@lists.sourceforge.net). The Snort community has used other means of communication since that time, including internet relay chat channels, a blog maintained by Sourcefire (the company founded by Roesch around the Snort program that was later acquired by Cisco Inc.), and three other listservs (*Snort-sigs*, *Snort-users*, and *Snort-openappid*). Nonetheless, Snort-devel has consistently been both the most used method and the officially sanctioned avenue for development-related communications within the FOSS community. Subscribing to the listserv, an act that is open to any interested party, grants an individual access to development-related messages and unofficially signifies his or her membership in the Snort FOSS community. The centrality of the Snort-devel listserv (hereinafter simply 'the listserv') to the Snort project cannot be understated. As such, the 15 years of archived messages held on the open source website Sourceforge are

an invaluable tool for this case study. Sourceforge, which is a website and data repository dedicated to FOSS projects of all stripes, is a well-known commodity in the FOSS community.¹ It has no affiliation with Sourcefire, and the similarity of the names is coincidental.

The cataloging of emails from the listserv is a data collection technique that mimics that used by von Krogh et al. (2003), who collected more than 11,000 emails from a similar archive of an open source project. The data for this current study begins with the first archived emails in August 2000 (when the listserv was created) and runs through the end of January 2014, a period of 13 years and 6 months. The end date was chosen for two reasons. First, in late 2013, Sourcefire was acquired by the much larger Cisco Inc. While all accounts indicate that Cisco has allowed its new subsidiary to continue operating its flagship product unhindered and without alteration, this acquisition may nevertheless have resulted in unforeseen changes in the governance of the project and the behavior of the community. Ending the data collection at this time eliminates such concerns and focuses the entire study on a period during which oversight and governance were consistent and predictable. Second, the end date coincided with the release of Snort v 2.9.6 on January 22, 2014 and trailed the late 2013 announcement of the alpha version of Snort 3.0 by only a few months. These two mileposts serve as useful cutoff dates, and the extension of data collection to a week past the release of a new version of the program ensures the inclusion of a period when activity on the listserv tends to spike.

¹ The Sourceforge Archive is available online at <http://sourceforge.net/p/snort/mailman/snort-devel/>. A separate archived copy of Sort-devel emails is maintained at <http://marc.info/?l=snort-devel>

In total, just under 9,500 email messages were harvested from the archive for the study time period. To transform these messages into usable data, the researcher coded each message to record information concerning both the message itself (including the main source of data for the dependent variable: Was the message identifying a bug in the program, contributing code, or suggesting a new feature to be added?) and its sender (including the main source of data for the independent variable, or the contributor's affiliation and nature of his or her contribution). Additionally, because the email messages are grouped into 'threads' (i.e., series of messages that are all replies to a single original email) in the archive, all these were coded in a similar fashion – which allows for individual ideas to be tracked in a given thread and thus provides insight into how ideas develop and are affected by input from the FOSS community. Just under 4000 threads were identified. Finally, all individual contributors were coded to track their affiliations, number of messages sent (as a measure of participation level), and number of contributions of various kinds; a total of 1543 unique contributors were observed. Of further help was the consistency of Snort-project and Sourcefire developers in using designated email addresses, which are easily identifiable by the respective domains @snort.org and @sourcefire.com. This norm allowed the data to be easily broken down into the aforementioned categories for the independent variable, namely *affiliates/employees* (who are not part of the greater FOSS community) and *non-affiliates* (i.e., the members of the greater community whose contribution to Snort are only made possible by the FOSS system). Identification, particularly of outside developers, was further assisted by information from email signature-blocks, which often share a contributor's employer or other affiliation.

To ensure internal validity, an inter-coder reliability check was conducted. In this validity check, two anonymous referees were each given an explanation of the coding process and a working review of the research and its goals before being asked to code a small subset of the data. The two referees were both familiar with software development and the FOSS community and had been given background information on this study. The results of these inter-coder checks (see Tables 3, 4, and 5) demonstrate high levels of agreement and sameness for the coding. Overall, 531 out of 591 data points were coded alike, for a total of 90% agreement.

Table 3: Inter-coder reliability check

	Percent Agreement	Scott's Pi	N Agreements	N Disagreements
<i>Referee 1</i>	91.065	0.623	265	26
<i>Referee 2</i>	88.7	0.610	266	34

While percent agreement is a useful measure for gaining a simple and readily understandable visualization of how similar two coders are to each other, this approach has limitations. The most concerning is the susceptibility that results are skewed by a single variable with low variance, which could mask the fact that other variables have much more problematic levels of variance.

One way to account for this challenge is to include a more descriptive measure, such as Scott's Pi, which takes into account the probability that data was coded alike by chance. The validity levels of Scott's Pi are known to be difficult to define, but because

the measure is somewhat more conservative than other similar measures in its definition of agreement, coefficients above 0.60 are generally considered good, while above 0.40 is also normally acceptable (Hallgren, 2012; Gwet, 2010). This 0.40 threshold is easily met by the data used in this dissertation.

Another approach to overcome the single low-variance variable is to break the data out into each variable and present the agreement results at that level. Variable level agreements are shown below in Tables 4 and 5, with Scott's Pi presented for each case.

Table 4: Inter-coder reliability check – Referee 1

	Percent Agreement	Scott's Pi	N Agreements	N Disagreements
<i>Variable 1 (Bug Contributions)</i>	90.722	0.613	88	9
<i>Variable 2 (Code Commits)</i>	88.659	0.554	86	11
<i>Variable 3 (New Feature Ideas)</i>	93.814	0.715	91	6

Table 5: Inter-coder reliability check – Referee 2

	Percent Agreement	Scott's Pi	N Agreements	N Disagreements
<i>Variable 1 (Bug Contributions)</i>	83	0.513	83	17
<i>Variable 2 (Code Commits)</i>	92	0.766	92	8
<i>Variable 3 (New Feature Ideas)</i>	91	0.421	91	9

In addition to analyzing the archived data, a series of ten interviews was conducted with long-time, significant contributors to the Snort project. Three of these individuals were Sourcefire employees who were interviewed in-person at the firm's facility in Maryland. The remaining seven were non-affiliated members of the FOSS community who had nonetheless been heavily involved in the development of the project over an extended period of time; these interviews were conducted via telephone. Guidance from Yin (2013) and Jorgenson (1989) was utilized to develop the model used for the interviews, which reviewed the Snort project in iterations or stages starting with background information and moving into a more detail discussion of intents, motives, and results.

The two main data sources (archive and interviews) were augmented by comments made in the text of the email messages themselves, publicly available documents on the development of Snort, such as the changelog (documents visible when downloading source code that note changes throughout the history of different versions) and other credits from within the source code, as well as a limited number of documents obtained directly from Sourcefire regarding Snort development. Taken together, this data is utilized to describe how the Snort team used external knowledge and took advantage of the FOSS community to build its product. This description can be generalized into a model of how such development might work in any similar software product.

CHAPTER FOUR: INNOVATION IN THE SNORT COMMUNITY

Introduction

By simple definition, an innovation is “the act or process of introducing new ideas, devices, or methods” (Merriam-Webster, 2016). Being innovative in this way is not stated as an explicit goal for either the Snort FOSS community or any FOSS community, at least in any recorded or noteworthy fashion. However, the goal of any given FOSS community undoubtedly *is* to build some software product and subsequently improve that product in pursuit of making it more useful. In the case of Snort, a problem was identified (i.e., people and organizations needed a way to monitor activity on their computer networks), and a new software program was written to solve it. Since the introduction of the product, new features and functions have been added continuously to improve the product and keep it relevant. By any definition, these additions to the software should be considered innovations – they represent new ideas, devices, and methods. In order to determine whether the FOSS community can produce superior innovation results, it is first necessary to understand how innovation works under such an arrangement.

Understanding innovation by itself, however, is not sufficient. It is also necessary to study how innovation occurs, or more accurately, what mechanism has enabled FOSS to produce superior results. This chapter examines how and why external knowledge flows from FOSS communities into software projects. It begins by presenting qualitative data that describes who contributes to the project and how, which reveals two major paths – direct input and indirect absorption, which can include the firm using community ideas or simply hiring talent from the community. This data is analyzed in conjunction with qualitative research, mainly interviews with contributors, to develop an understanding of

two major paths via which external knowledge enters the project. These paths are discussed in turn.

The Snort Open Source Community

In the more than 13.5 years from the beginning of the Snort FOSS email listserv in August 2000 until the release of Snort v 2.9.6 on January 23, 2014, some 8891 total messages were exchanged using the Snort developers mailing list – a rate of just under 2 messages per day. This activity represents the overwhelming majority of the external knowledge committed to Snort from its' community. The total figure does not include obvious spam, duplicate messages, or messages sent in error, which would bring the overall message total to nearly 10,000. During this time period, 785 bugs were submitted via the listserv, 755 ideas for new features for the product were suggested, and new code was offered in 1080 different messages. Collectively, these bugs, new features, and code commits represent the sum total of external knowledge contributed by the community – the dependent variable in this study.

Complete summary statistics for the listserv archive are presented below in Table 6. Some clear trends are immediately obvious from the data. First, a notable discrepancy between mean and median values exists across the entire dataset. This is not surprising given that of the 1542 total participants, 761 (49%) posted only a single message. That same subset of participants also provided only 14% of the bug reports, 11% of the new

Table 6: Summary statistics for Snort-devel email listserv archive

		All Contributors	Sourcefire Employees Only	Community Members Only	Code Submitters Only	Feature Submitters Only
Messages	Total	8891	2241	6650	6112	5563
	Mean	5.76	58.97	4.42	15.83	16.22
	Median	2	8	1	3	4
Days Active	Total	-	-	-	-	-
	Mean	174.14	1013.61	152.94	440.21	428.35
	Median	2	528.5	1	39	55
Bugs Submitted	Total	785	97	688	646	465
	Mean	0.51	2.56	0.46	1.67	1.36
	Median	0	0	0	1	0
Code Submitted	Total	1080	200	880	1080	865
	Mean	0.7	5.26	0.58	2.80	2.52
	Median	0	0	0	1	1
Features Submitted	Total	755	92	663	623	755
	Mean	0.49	2.42	0.44	1.61	2.20
	Median	0	0	0	1	1

feature ideas, and 10% of the new code contributions. Relative to the entire community, this large group of participants thus has little to no engagement in the daily ebb and flow of project development. While this is apparent when examining a robust archive such as the listserv, the very existence of this group may go largely unnoticed by the project's actual participants. This may be because serious participants are inclined to only engage with community members who display a sufficient interest or whose contributions are 'up to par.' These community members subsequently become part of both the project and its culture to the extent that the phrase 'like family' is not an exaggeration. Consider the words of one long-time Sourcefire employee:

We haven't had a lot of ebb and flow. We have a lot of people that are just constant for a long time. We've been around long enough to where we've had several prominent people in our community, you know, die. It's very sad, but...Yeah, I can think of 3 or 4 people we've lost that were very

active and very vocal in the community, help people out, and helped each other out, this that and the other thing

The ‘we haven’t had a lot of ebb and flow’ remark may at first appear to contradict the data, but this is not an isolated viewpoint. Matt Mickel, a Sourcefire employee who spends a great deal of time working with the community, observed:

We had one gentleman that was very active that moved on to, I think the story is that his situation changed in where he was living where he was working, something along those lines changed. He was a very active submitter and has significantly dropped off...And someone like him will disappear for a while and other people will start submitting things and it’s kind of neat to see that if someone new starts submitting things, it’s either a rule or a signature or a signature being added to the list, it’s exciting for them, and they want to start submitting more.

The focus here is clearly on the quality of participation in the community, not on the quantity. In Mickel’s view, the loss of a single community member who was a significant contributor stood out as noteworthy. At the same time, however, countless others come and go from the community with hardly a notice from the core members, as they have made little to no contribution. The remarks regarding ebb and flow therefore reflect an assessment of only those community members who are core contributors, not of the entire community.

Additional trends are observable when the data is broken down by affiliation and behavior. For example, employees of Sourcefire (and later Cisco Inc.) have more active and lasting engagement on the listserv. While non-employees are active for an average of 153 days and post on average 4.4 messages over the course of their participation, the respective numbers for employees are 1014 days and 59.0 messages. The level of activity

by employees is not surprising, as many Sourcefire employees have interacting with the community via the listserv as part of their job description.

More interesting is the apparent correlation between actively contributing to the project in some way and higher levels of participation in the discussion. Compared to the baseline of all community members (who are active for 174 days and send an average of just 5.76 messages via the listserv), members who submit new code contributions for the project are active for 440 days and send an average of 15.83 messages. Community members who post ideas for new features to be added to the project are even more active, with an average of 16.22 messages sent – although they do so over a smaller period of time than code contributors (namely just 428 days).

These numbers describe something about the community: being actively engaged is important. Moreover, the ranks of those who are actively engaged differ significantly from those of the community as a whole. While a total of 1543 individuals contributed to the project during the study period, only 386 contributed code and 343 contributed new feature ideas. Allowing for the fact that these subsets have some overlap (i.e., some individuals contribute both code and new feature ideas), a total of 509 individuals engaged in either one activity or the other. This means that less than one third of all participants (32.99%) accounted for *all* of the code and new features brought into the project by the community. As a result, it is easy to see why those at the core of the community may dismiss out-of-hand the fleeting presence of the other two-thirds, whose effect on the project is either cursory or completely null.

Beyond creating an understanding of how community members view one another and the relative importance of different participants, this data provides an opportunity to

identify how the FOSS approach has expanded the body of knowledge related to Snort that could potentially contribute to innovation. One way to understand this is to compare Snort's community to that of a proprietary software development team. For example, in a 2008 blog post, Steve Sinofsky, who was then in charge of product development for Windows 7 at Microsoft, noted that "the Windows 7 engineering team is made up of about 25 different feature teams...on average a feature team is about 40 developers, but there are a variety of team sizes" (Sinofsky, 2008). This means that Microsoft had approximately 1000 developers working on its flagship product. By comparison, even today – when Sourcefire is much larger than it was in 2008 – this firm has only 40-50 developers working on Snort. Nonetheless, leveraging 509 FOSS contributors enables Sourcefire to gain a tremendous advantage in the size and diversity of its team. A complete analysis would correctly account for the reality that not all of Snort's 509 contributors are active at the same time and many of these individuals split their attention among many different endeavors. However, it would also reveal that in 2012 (the last year for which data on both firms is available), Sourcefire had \$223.1 million in revenue while Microsoft had \$73.72 billion.

Direct Input: Community-Driven Innovation

Now that a description of what the Snort FOSS community looks like and how it compares to the more traditional software development community has been presented, the question of what this data says about how the FOSS community contributes to innovation arises. Clearly differences exist between the behaviors of Sourcefire/Cisco employees and the rest of the community. But does this affect how development is undertaken? What can the email archive reveal about how external knowledge flows in and out of the Snort community?

The data describes how many innovative new ideas are contributed to the project by the community, as well as the manner in which the new code that is needed to implement those ideas has been brought in. It can even provide some limited information about the affiliations and behaviors of the contributors of these ideas, such as whether they are working for another firm or are frequent contributors. In truth, however, the data alone only tells a part of the story. It does not describe motivations (i.e., why these individuals contribute), although this is only tangential to the main question posed by this research. More importantly, the data does not fully describe how the flow of external knowledge into and out of the Snort project, courtesy of the FOSS community, facilitates greater innovation. Merely having access to more programmers than the project would have otherwise is certainly a good starting point, as a larger body of knowledge will presumably raise the *potential* for innovation; however, it does not guarantee success.

The discussions that actually take place in the email messages exchanged through the listserv add invaluable context to the story begun by the raw data. In particular, they reveal a diverse set of motivations for contributing, a wide range of ways in which to do so, and a dynamic of contribution and participatory enthusiasm. These interactions turn the potential of greater innovation into real results within the project.

Consider an early example of an outsider coming to the Snort FOSS community (see Figure 1 below) with an idea for a new feature to be added to the product (Sourceforge Archives, Snort-Devel Listserv 2015). Several things stand out about this message. First, the sender, a non-employee community member, explicitly states a motivation for working on the new plugin. His motive appears to be simply a desire to see an improvement added to Snort. This is a telling clue about what moves the

community to action and indicates that simple intellectual curiosity (meaning a desire to pursue one's own line of inquiry) plays a large role in the FOSS community.

```
[Snort-devel] new plugin for rules based on stream offset
From: scott campbell <axonpotential@ya...> - 2002-07-30 00:43:55

I put together a plugin this weekend which will allow
you to associate snort rules based on stream offset
(in packets or bytes). I was thinking this might be
useful for gathering ssh version strings, without the
overhead of trying to string match the entire rest of
the session.

There is one problem - although the logic is being
thrown when the appropriate number of packets/bytes
is being seen (ie the test function sees the correct
return value), the logging is inconsistent. Some
functions like the packet # < always log, while none
of the others do.

I am out of ideas and looking for feedback.

I set up a link on the axonpotential site with the
source code, example rules and a little more
information.

http://www.geocities.com/axonpotential/snort/sp\_stream/index.html

thanks!

scott
```

Figure 1: Snort-devel listserv conversation 1

In the non-FOSS setting, such activity would not be possible, as source code is strongly protected and not available to the general public. In short, this new knowledge would not have been brought into Snort if it were not FOSS. Furthermore, the originator of the message is actively soliciting input and feedback from the rest of the community – which he subsequently receives in the form of five responses from two different community members, one a Sourcefire employee and one another non-employee (see Figure 2).

Re: [Snort-devel] new plugin for rules based on stream offset

From: scott campbell <axonpotential@ya...> - 2002-07-30 23:03:36


```
I just wanted to say thanks for the help on this one.  
By using todays cvs and commenting out the  
PKT_REBUILT_STREAM test it all seems to be working  
perfectly now.  
  
I will do a bit more clean up and then post it.  
  
Chris - do you want me to change it to  
'stream_depth: '? If it looks good it's all yours, but  
if you all had something else in mind to supersede  
this I would not take it personal...  
  
Another day or so and I will do a formal release  
posting.  
  
thanks again to both of you!  
  
scott
```

Figure 2: Snort-devel listserv conversation 2

The three individuals involved in this discussion together sort the initial challenges out, with the two respondents both suggesting new lines of code until the issues are all resolved. The ‘Chris’ mentioned in this last message is Chris Green, a Sourcefire employee who is participating in the discussion. Green acknowledges the success of this collaboration several days later in a final message confirming “Going to do this then do a beta release” (Sourceforge Archives, Snort-Devel Listserv). A new idea presented by an outsider has thus found its way into Snort in only six short days, which demonstrates the direct link between external knowledge and innovation.

In other cases, development works in nearly the opposite fashion, with Sourcefire employees providing suggestions that are then taken up by community members. In one such example, a community member posted a code patch for a new Snort feature on

December 25, 2008. Martin Roesch, Sourcefire's founder and chief technical officer (CTO), responded 11 days later (Sourceforge Archives, Snort-Devel Listserv), as show in Figure 3 below.



Re: [Snort-devel] Patch for snort 2.8.3.1 - Decompressing Gzipped HTTP responses
From: Martin Roesch <roesch@so...> - 2009-01-05 21:29:36
Attachments: Message as HTML

Hi Breno,

This is really cool, thanks for sending it in. I have some comments.

Figure 3: Snort-devel listserv conversation 3

Roesch then went on to list four suggestions for improving the patch. The original sender replied: “Thanks for all the suggestions. I will work as soon as possible in the code and fix it...I’ll send you back soon!” (Sourceforge Archives, Snort-Devel Listserv). Updated code was posted the next day, and the feature was implemented into Snort. The interaction between the Sourcefire employee and the external knowledge contributor was again key to this new addition.

Motivations for contributing to an open source project are covered in a large body of research. According to the findings, possible forces at work include prestige, reputation, self-satisfaction, an indirect means to profit, and employment concerns. However, this research is most useful for understanding the causes of FOSS community formation (i.e., why individuals join the community). For the purpose of understanding the effects of community formation – specifically the effects on innovation – questions of

motivation are secondary to questions of efficacy. The FOSS community is clearly effective in introducing new external knowledge into Snort. In addition to providing a means for previously disconnected knowledge to come together, the community also provides a platform for active discussion and collaboration. As a result, external knowledge is not only made passively available to Snort's developers; it is also actively integrated into the product. Internal development teams in proprietary development environments may use the same tools (e.g., discussion boards, chats, and email lists) to communicate internally, but what sets the FOSS community apart is its ability to easily interact with external knowledge, which is an asset that simply does not exist in the non-FOSS world.

Shifts in Community Behavior and Makeup

In considering the direct contributions of the community to the Snort project, one over-arching trend warrants further discussion: a notable decline in message activity over time. In 2001, the first full year of the listserv, more than 2000 email messages were exchanged; by early in the next decade (namely 2011–2014), that number averaged less than 500 per year. According to Sourcefire employees and members of the Snort FOSS community, that was a result of the program's gradual maturation over time, along with a growth in the role of Sourcefire as the leader of Snort development. In short, increasingly fewer ways to improve the product are available over time, while a larger number of able minds are working at Sourcefire to take on a greater share of the work.

Generally speaking, Snort community members who were asked as part of this research suggested that a period of initial activity (during which a great deal of work was done to build the program from the ground up) was followed by a steep decline in activity that eventually reached a stable 'maintenance' level that can be expected to continue for

some time (see Figure 4). The suggestion that contributions from the community have dropped over time is borne out by evidence from the listserv. For example, measuring messages per year reveals that the predicted drop and leveling of activity are confirmed by the data, as seen in Figure 5.

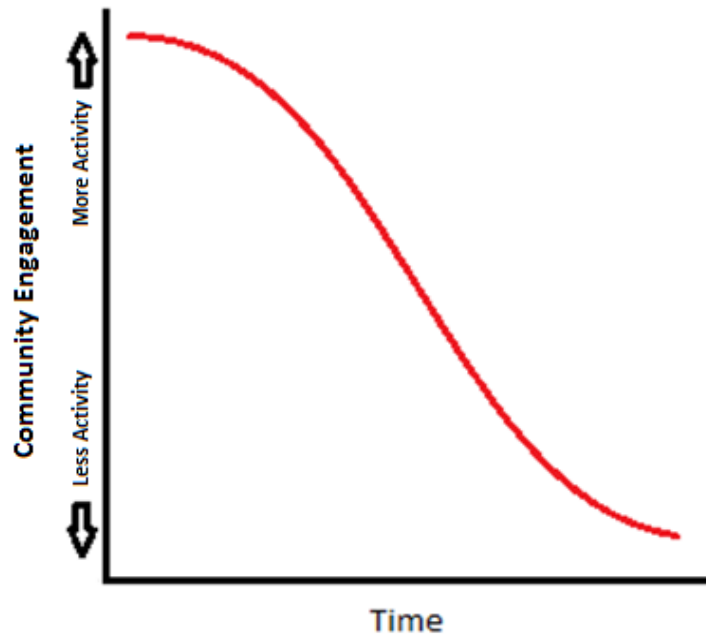


Figure 4: Predicted Snort FOSS community activity

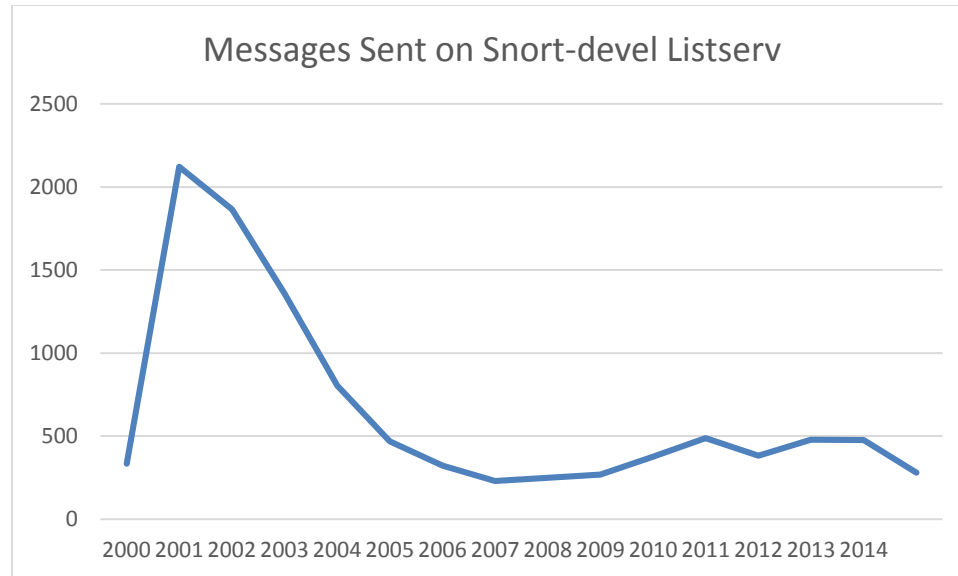


Figure 5: Actual Snort FOSS community activity

It is not just activity that is decreasing; contributions derived from the community appear to be falling over time as well. Figures 6, 7, and 8 represent contributions of each innovative input from non-affiliated community members over time. Each measure reflects marked decreases in the rates at which the community has been offering the three types of contributions. For example, as demonstrated in Figure 6, bug contributions from the community were 233 in 2001 and 174 in 2002 – but then fell sharply to 23 in 2011, 17 in 2012, and 45 in 2013. Once again, Sourcefire officials and those in the community both expected such a trend. According to Joel Esler, Sourcefire’s open source community manager,

Because Snort is as old as it is at this point, it’s you know, 15 plus years old, it’s extremely stable so we don’t have a lot of code contribution any more”. Similarly, one FOSS community member remarked on open source projects in general “(it) depends on the project. More active projects...get

more support from the user community, projects like libpcap/tcpdump much less so, since they're pretty mature at this point.

These numbers tell a compelling story of a decline in the FOSS community over time. However, even those who are aware of this shift and highlight it in comparison to the rising role of the firm in developing Snort are careful to emphasize that this decline in activity does not equate to a decline in the importance of the community. When asked to describe the ratio of lines of code contributed by the community vs. those developed in-house, Esler stated clearly that some 99% of the code for Snort is now written in-house. This estimation is supported by the data in Figure 8, which reflects a dramatic drop in the rate of code contributions from the community – during a time in which Sourcefire was rapidly growing and undoubtedly adding internal capacity. However, Esler goes on to state during interview that the community continues to be “both strategic and it’s helpful. It helps us a whole lot. We can put stuff out and test it. We can say ‘Hey, we’d like for you guys to test these things. Can you test these things?’ Right, we do alpha, we do beta releases of our code before we put it out stable you know, and we do that quite often.”

It is clear that ideas generated in the community can have an impact on the product/project that is the focus of FOSS activity, in this case Snort. However, this last revelation from Esler suggests that ideas that are generated in the smaller group that runs the project (here the software firm Sourcefire) can also involve the greater community, even though they are not developed there. While it is easy to conceptualize that an idea can originate in either one place or the other (that is, internally within Sourcefire or externally in the greater community), it is less simple to describe how the nature of the community will affect ideas that are internally generated. This second idea is important,

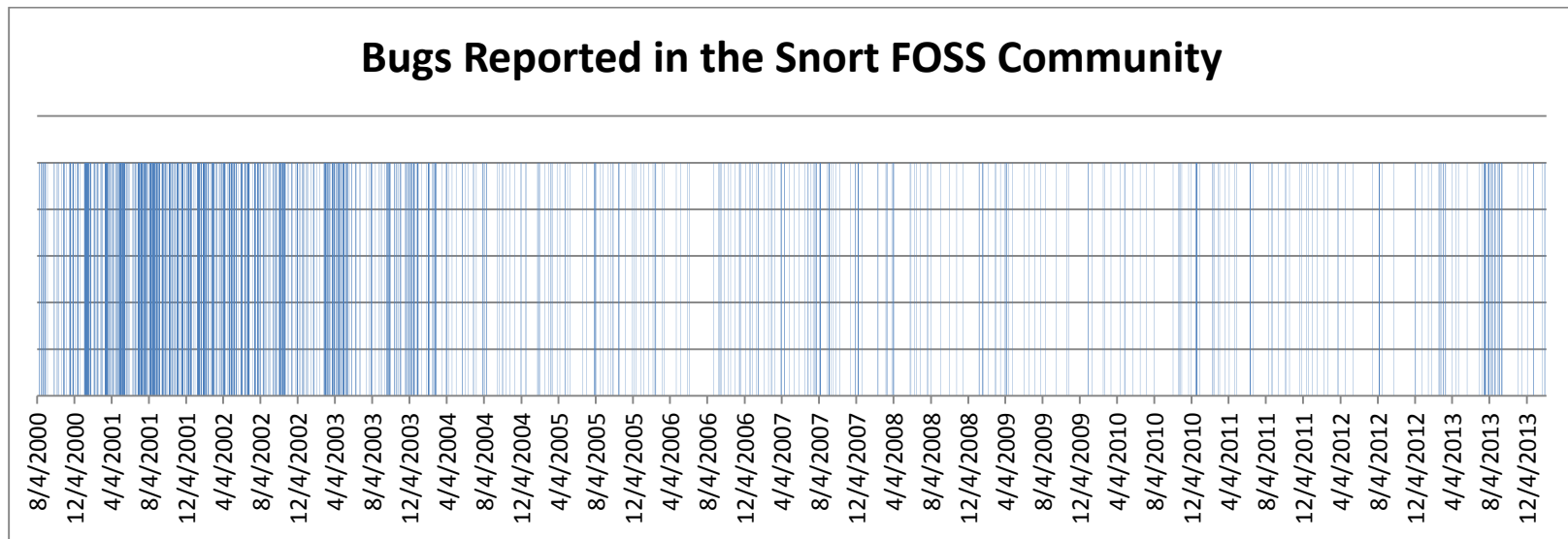


Figure 6: Bug submissions to the email listserv over time

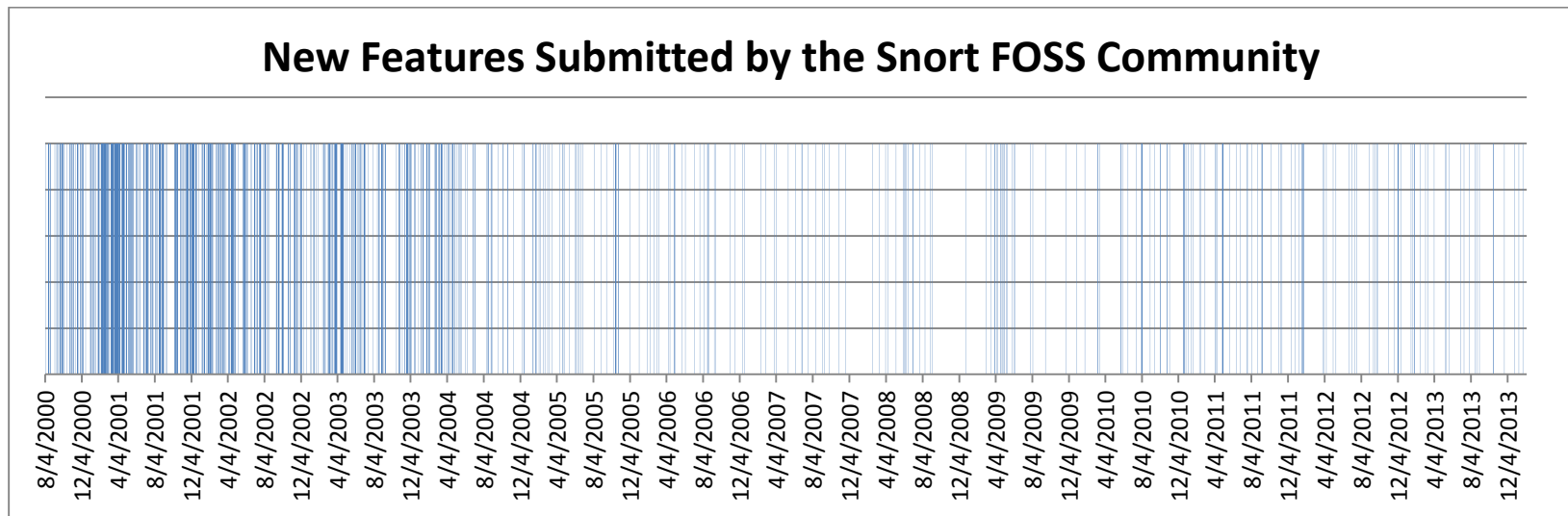


Figure 7: Feature submissions to the email listserv over time

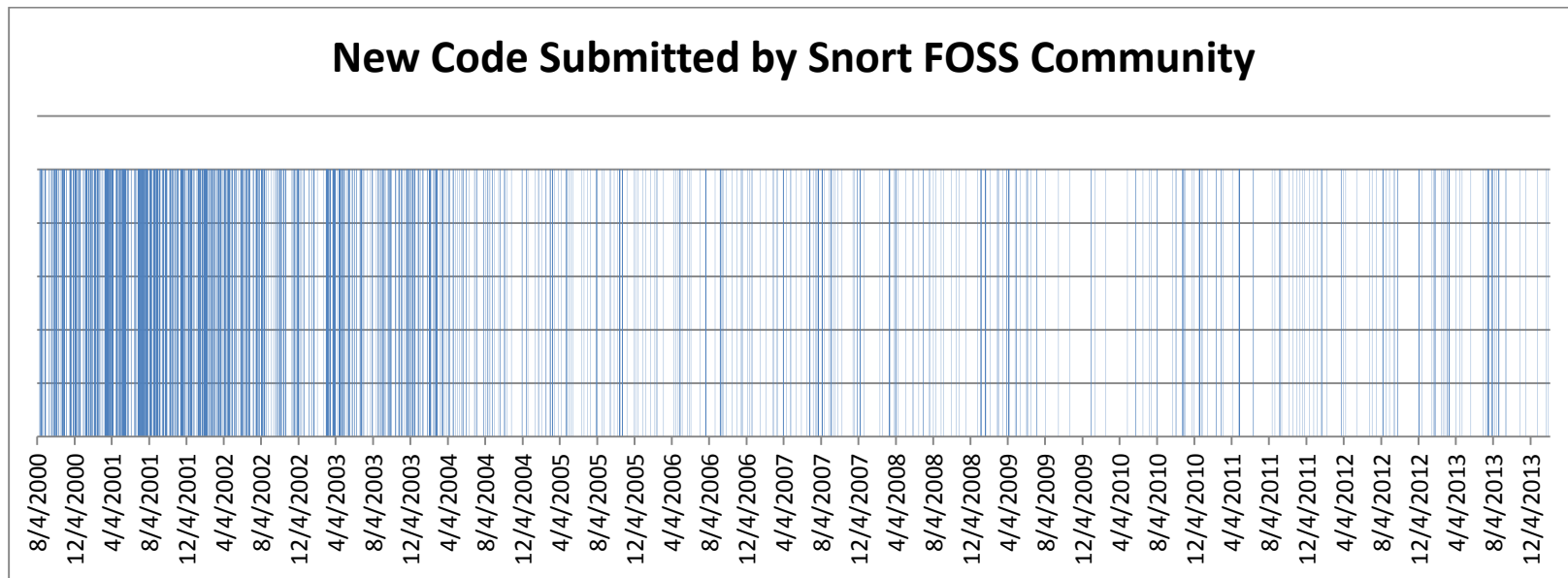


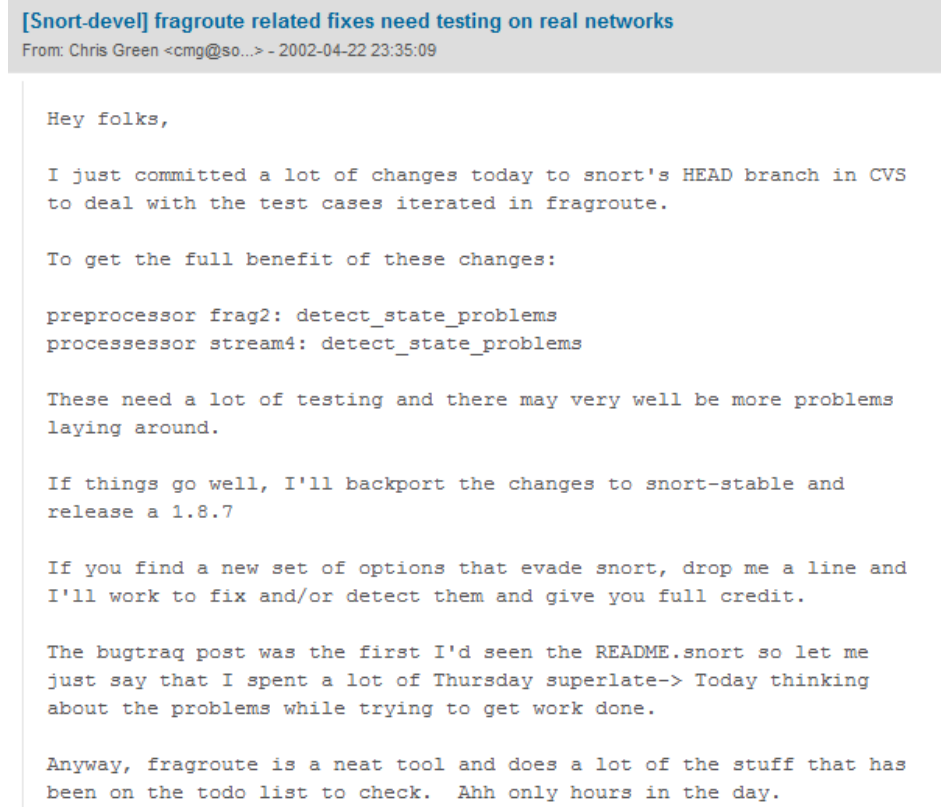
Figure 8: Code submissions to the email listserv over time

particularly in light of the demonstrated decline in community contributions over time.

Understanding this second nature helps to explain if the community's role is diminishing or simply evolving as the project matures.

Knowledge Absorption: Community Contributions to Internal Innovation

The common use of the listserv by all Snort developers provides helpful insight into the process of community involvement with internal development. For example, in 2002 an email thread was begun by a Sourcefire employee to note that a new tool had been added to Snort (see Figure 9).

The image is a screenshot of an email thread from a listserv. The header shows the subject as "[Snort-devel] fragroute related fixes need testing on real networks" and the sender as "From: Chris Green <cmg@so...> - 2002-04-22 23:35:09". The body of the email is formatted as plain text and contains the following content:

Hey folks,

I just committed a lot of changes today to snort's HEAD branch in CVS to deal with the test cases iterated in fragroute.

To get the full benefit of these changes:

```
preprocessor frag2: detect_state_problems
processsor stream4: detect_state_problems
```

These need a lot of testing and there may very well be more problems laying around.

If things go well, I'll backport the changes to snort-stable and release a 1.8.7

If you find a new set of options that evade snort, drop me a line and I'll work to fix and/or detect them and give you full credit.

The bugtraq post was the first I'd seen the README.snort so let me just say that I spent a lot of Thursday superlate-> Today thinking about the problems while trying to get work done.

Anyway, fragroute is a neat tool and does a lot of the stuff that has been on the todo list to check. Ahh only hours in the day.

Figure 9: Snort-devel listserv conversation 4

This initial solicitation was followed by a chain of five messages from three different individuals over the course of 24 hours, with an allusion to at least one additional private message regarding the same subject. In this discussion, the original sender accepted input from two FOSS community members, neither of whom was a Sourcefire employee and added their input into the code supporting the new tool: “Ok, just made a change based on feedback from Andrea Barisani and Peter Johnson to actively alert on overlapping fragments and it is in both stable and devel” (Sourceforge Archives, Snort-Devel Listserv 2015). This is clear evidence that even when an idea originates internally, ample opportunity for the greater community to contribute its knowledge in an effort to enhance the product still exists. In short, external knowledge is still being leveraged in support of innovative activity.

Using the FOSS community to ‘bounce ideas off of’ is a consistent hallmark of the Snort listserv. While this activity often takes the form of the previous example (in which a new feature was introduced and the community offered unsolicited commentary), the community is just as often approached directly by Sourcefire for feedback, as shown in the message presented in Figure 10.

[Snort-devel] New Proposed Classification.config file setup

From: Joel Esler <jesler@so...> - 2010-12-23 22:27:19

Attachments: [Message as HTML](#)

As mentioned earlier, here's the proposed Classification.config file setup posted and available for download here:

<http://blog.snort.org/2010/12/new-proposed-classificationconfig-file.html>

Please take a look, leave comments preferably on the blog, but also here would be fine.

I'll leave it open for comments until January 12. At which time I'll assemble the comments into a final product for inclusion into Snort.

Thanks for your support, and thanks to the Snort/Emerging Threats community for the idea.

Joel Esler
Manager, OpenSource Community, Sourcefire

Figure 10: Snort-devel listserv conversation 5

Sourcefire's rather direct and overt solicitation of input from the FOSS community here is a powerful illustration of the role that community plays. This initial posting set off a 15-message exchange that spanned the course of 89 days (it is noteworthy that the final message was sent on March 22, 2011 – well after the January 12 deadline for input imposed by the original poster). A total of ten individuals participated, including six community members and four Sourcefire employees. Code was contributed by three of the community members and one employee. One of the participants worked for the Gentoo Foundation (an open source group), while another worked for the computer security firm netpublishing.com (Sourceforge Archives, Snort-Devel Listserv). It is difficult to imagine the end product being the same had these community members not participated in its development. It is similarly difficult to envision a scenario in which a proprietary (i.e., non-FOSS) software development firm or group would be able to directly solicit this type of external knowledge so quickly and efficiently.

These types of discussions permeate the listserv. In a message from May 2009, Snort creator Martin Roesch offered to the community:

I wrote a patch for Snort 2.8.4.1 that implements IP blacklisting as a preprocessor in Snort over this past weekend. We talked about this last week on the mailing list in regards to trying to implement blacklisting using regular Snort rules and how well that doesn't work...Have a look and let me know what features you'd like or bugs you find. This code is purely EXPERIMENTAL (Sourceforge Archives, Snort-Devel Listserv).

That the program's creator would seek external input for a new patch indicates the important role ascribed to community knowledge.

In fact, the community can even unintentionally provide the spark for an idea that is later implemented internally. An email thread started in August 2004 when a community member posed a question about how to use a particular Snort database. This query sparked an eight-day exchange between six developers – two Sourcefire employees and four participants from the community. In the course of this discussion, which involved nine messages in total, one of the Sourcefire employees came up with an idea to improve the relevant part of the program. After this employee introduced the idea and received feedback from at least two community members, code was offered and the idea implemented (Sourceforge Archives, Snort-Devel Listserv).

Beyond demonstrating the reach of external knowledge, these exchanges also yield clues as to how innovation works in the FOSS community by demonstrating an iterative process that is both collegial and communal in nature. This should perhaps not be surprising given the nature of the FOSS community, which tends to be made up of like-minded and similarly skilled individuals. Nonetheless, the fact that a firm like Sourcefire, which has considerable intellectual capital of its own, would actively seek

input from the community even when it does not have to shows the important role of this community.

Talent Acquisition by the Firm

While it is clearly possible for community contributions to play a role in the development of FOSS even as a particular project matures, small contributions to development discussions may be outweighed at some point in the maturation process by the work of in-house developers. This is an inevitable state of affairs if, as suggested previously, 99% of Snort's development is now done in-house. However, this reality again does not mean that the utility of the community has degraded. This may not seem obvious at first, but the explanation for why this is the case can be revealed by answering the question: If most development is done in-house, where does Sourcefire find its in-house talent?

As may be expected, the answer is that Sourcefire relies heavily – indeed, almost exclusively – on the FOSS community to identify and recruit talent. As Joel Esler states rather bluntly: “That’s how we built the company. We hire our community. That’s how we built the company.” Naturally, the process and purpose are both somewhat more complicated than this initial statement, although it still illustrates the importance of the community in building a firm that relies so heavily on a FOSS product. As Esler further elaborates:

It’s a lot about, you know, we identified a lot of good people in our community. These people are very smart. These people are good at writing code...And that’s how SourceFire started in Marty’s Living Room, is he hired the people that were working on Snort at the time. You know, people that were contributing code to the open source community, he hired four or five people that were contributors. And a lot of our people are out of that community

This approach of hiring talent from the community is visible in the listserv. Several participants who were active members of the community eventually began sending messages from new email addresses with @sourcefire.com domains, which confirmed that they had moved from being community members at large to being Sourcefire employees. Some stayed on throughout the duration of this study, while others left after a short time. One left and then returned to Sourcefire for a second stint. Others in the community have been offered jobs by the firm but turned them down, preferring to stay where they are and to continue to contribute only as a community member. Two of the community members who were interviewed for this research confirmed that Sourcefire had offered them jobs in the past.

Identifying talented code-writers is only one advantage of using the community to build a workforce. Familiarity with the program is clearly a tremendous advantage for both a job candidate and the hiring firm. Sourcefire managers such as Mickel and Esler stress this as a reason for focusing hiring efforts on the community – individuals gain experience using the program in their current jobs and begin contributing as community members, which demonstrates their capabilities to Sourcefire. According to the same Sourcefire officials, however, the ability to find potential employees who fit into Sourcefire's culture is equally important. In that regard, participation in the FOSS community serves as a kind of weeding out process through which certain individuals become part of a group that embraces the open source ethos that is at the core of Sourcefire's company culture.

When asked to describe the culture and mentality of Sourcefire, Esler responded as follows:

To work in this group you have to think differently. You have to think....it's not really about thinking outside the box. There is no box...And we hire based off of that. Right. I've said to a lot of interview candidates and I've said to a lot of people that want to be interview candidates to come here...it's not about your technical skill. If you're smart and you can figure out how to do what we're going to teach you to do, we'll teach you how to do your job. It's how you think.

As a firm, Sourcefire clearly believes that the type of thinking described above goes hand-in-hand with being a part of the FOSS community. While community membership or even working with Snort in the past are not official pre-requisites for employment with Sourcefire, they serve as de facto gateways into the company. In turn, the company's innovative engine was built using parts sourced from the FOSS community.

Conclusion

The Snort project captures the benefits of external knowledge in many ways. Direct input and knowledge absorption (both via internal use of community knowledge and the acquisition of talent by Sourcefire) are two unique paths that can lead external resources from the FOSS community to Snort. The connection to innovation then is obvious: with significantly greater knowledge resources due to the size of its FOSS community, the Snort team gains the potential to achieve better outcomes than peer products. Furthermore, by finding particular ways to merge that external knowledge with its existing internal stores of knowledge, the Snort team ensures that this potential is realized in the form of a product that is unique, more capable, and more innovative than it

would be otherwise. Even though the people who run Snort assert that greater innovation is not defined as a goal of either the project or its FOSS orientation, such a result is nonetheless a byproduct of the choice made to build Snort in the FOSS community. This is because while innovation is not a core FOSS goal, a willingness to accept external input (i.e., external knowledge) is in fact a fundamental part of the FOSS ethos.

CHAPTER FIVE: OPEN SOURCE, INNOVATION, AND SECURITY

Introduction

This chapter builds on the previous chapter by examining a unique structure that has emerged within the Snort project, namely a sort of ‘community of interest’ that is focused around contributing to the security of the software code that constitutes Snort. This community represents a body of knowledge and a subset of experts within the FOSS community that can describe a great deal about how and why external knowledge comes into the system. Prior to this analysis, a groundwork of understanding about software security, what constitutes a security bug, and why the bug is important is offered. This chapter concludes with a discussion of the importance of the community of interest and the ways in which it enables the two major types of knowledge contributions to the project (as described in Chapter Four).

Software Security

Although cybersecurity is a concept that is widely misunderstood by the general public – and even by many policy makers – the challenge associated with it is steadily gaining prominence in the minds of both groups. At the level of national security policy, the first steps toward a comprehensive approach for addressing this challenge were taken in 2008 with the signing of National Security Presidential Directive 54/Homeland Security Presidential Directive 23, which is known as the Comprehensive National Cybersecurity Initiative (CNCI). This since-declassified directive states, in part, that “The electronic information infrastructure of the United States is subject to constant intrusion by adversaries that may include foreign intelligence and military services, organized criminal groups, and terrorists... These activities cost American citizens and businesses

tens of billions of dollars each year” (FAS, 2016). The CNCI was taken up by the Obama administration, which expanded the effort to include a comprehensive cybersecurity policy review that resulted in the foundation of the current U.S. cybersecurity policy. A host of business, industry, and other private-sector actors also have significant interests in good cybersecurity policies and practices. This is no surprise, given the financial stakes of cybercrime and other nefarious online activities. In 2014, Intel Security (formerly McAfee) and the Center for Strategic and International Studies estimated that costs to the global economy resulting from cybercrime are as much as \$575 billion annually, with \$100 billion in costs in the United States alone (Kopan, 2014). According to another report, annual costs could rise to more than \$2 trillion globally by 2021 (Juniper, 2015).

The software security sub-field of cybersecurity, along with more regular concerns over vulnerabilities in otherwise useful and widely adopted software programs, occupy prominent roles in an ongoing clash between cybercriminals and IT/computer/network security professionals. Bugs are often responsible for significant cybersecurity failures. For example, the much-publicized Shellshock bug was not a virus or other malicious piece of code, but rather a mistake in the actual code of a widely used Unix bash shell – which illustrates just how critical such bugs are to software. To exacerbate the issue, software bugs often exist for months or years before being noticed by security professionals. It often occurs that no one is aware of a bug until it is too late – some bad actor has taken advantage of the situation and the damage has been done. Producing software of the highest quality is therefore an imperative within the industry and a task that figures prominently in the minds of developers. Many of these developers – including those involved in Snort and other FOSS projects – feel that secure software is

just as important as functional software. Finding new ways to make their products secure is just as innovative a pursuit as adding new features that enhance a program's functionality.

Open Source-Fueled Security and Innovation

When asked how Snort's open source nature affected innovation in the end product, Joel Esler, the Sourcefire open source community manager, responded "people will find a bug and generally when they find a bug, they submit the patch to fix the bug with it...and we pride ourselves very heavily on the ability to react faster. So if you have a security problem and we patch it, we can release a new version...the next day." In response to a similar question, Bill Parker, a Snort FOSS community member who is not employed by Sourcefire, stated: "finding bugs/issues is definitely easier, but this all depends on the project and how fast they can get the bugs fixed." When asked about improved innovation, those involved in building Snort consistently chose to reply by describing the extent to which open source helps to make a more secure product. That these software developers reply to a question about innovation with an answer about security strongly suggests that, at least in the minds of those who work on the project, the two concepts are inextricably linked. That link in turn has significant implications for FOSS, innovation, and security applications.

In comparing the security of FOSS products to their non-FOSS peers, Hoepman and Jacobs (2007) apply the highly useful analogy of a locksmith. Is it better, they ask, to trust securing your home to a locksmith who carefully guards his or her trade secrets so that no one else, thief or otherwise, knows how the locksmith secures a door? Or should more trust be placed in a locksmith who freely shares trade secrets with the world so that all manner of thieves, fellow locksmiths, and laypersons may examine the locks for

design flaws and other vulnerabilities? The debate is understood as *security through obscurity* vs. *security through transparency*. Hoepman and Jacobs (2007) conclude that the transparency approach (i.e., open source software) is more secure for two reasons:

1. With closed source systems, the perceived exposure may appear to be low, while the actual exposure (due to the increased knowledge of the attackers) may be much higher; and
2. Because the source is open, all interested parties can assess the exposure of a system, hunt for bugs and issue patches for them, or otherwise increase the system's security. Security fixes become available quickly, which means the period of increased exposure is short.

These conclusions are but the most recent in a significant body of literature demonstrating that FOSS products are more secure than peer products that do not take advantage of the FOSS commons.

A more empirical analysis is offered by Witten et al. (2001), who examined the average number of days between public announcements of discovered security flaws in competing FOSS and non-FOSS programs. They found that errors were discovered and reported much more frequently in open source products, which reflects the quicker detection of such errors. In discussing their findings, the authors cite several possible drivers of this discrepancy, from short lifecycles in between releases for open source products to the possible disincentives for allocating more resources to security that profit-seeking firms face.

Despite a growing body of research demonstrating that FOSS may have advantages over other software when it comes to security, those who believe that the *security through transparency* model fails by providing malicious actors too great an access to source code still have some doubts. According to this argument, FOSS is actually less secure. Hansen et al. (2002) explored this question in a study of FOSS code. However, while they note that room for improvement exists and some measures could be taken to improve security and trustworthiness, the researchers conclude that “Open Source is no panacea for security problems, but can give a big boost to trustworthiness...As has been proved in many Open Source projects, an open and co-operative software development can lead to robust and reliable products” (Hansen et al. 2002). Subsequent discussion has further moved the needle in favor of the benefits of FOSS outweighing its risks. In 2015, Jim Fruchterman, founder of software firm Benetech, summed up these feelings as follows: “With greater transparency, accountability, independent verifiability, and collaboration comes stronger security” (Fruchterman, 2015).

Bowing to both the sentiments expressed in the research cited above and the rising tide of pro-FOSS attitudes in the information security community, security-minded organizations have not only been accepting the use of FOSS in recent years; they are also embracing it. For example, the U.S. Government, and in particular the Department of Defense (DoD), are among the most interested parties in network security internationally. In 2003, the DoD commissioned the cybersecurity firm MITRE Corp. to investigate the role of FOSS in securing DoD networks. This firm found that, in 2003, “FOSS software

plays a more critical role in the DoD than has generally been recognized” (MITRE, 2003). The report went on to note the following:

One unexpected result was the degree to which Security depends on FOSS. Banning FOSS would remove certain types of infrastructure components (e.g., OpenBSD) that currently help support network security. It would also limit DoD access to and overall expertise in the use of powerful FOSS analysis and detection applications that hostile groups could use to help stage cyberattacks. Finally, it would remove the demonstrated ability of FOSS applications to be updated rapidly in response to new types of cyberattack . Taken together, these factors imply that banning FOSS would have immediate, broad, and strongly negative impacts on the ability of many sensitive and security-focused DoD groups to defend against cyberattacks... Neither the survey nor the analysis supports the premise that banning or seriously restricting FOSS would benefit DoD security or defensive capabilities. To the contrary, the combination of an ambiguous status and largely ungrounded fears that it cannot be used with other types of software are keeping FOSS from reaching optimal levels of use. (MITRE, 2003)

Far from indicted FOSS, the report emphasized the important role that FOSS could play in the future of network security. The emphasis on the utility of FOSS and the attendant implication that products built in this open knowledge community are more secure is an important observation for the current study.

Security Fixes as Innovation

It is challenging to understand security as an innovation. While the traditional view of innovation implies a positive addition (namely bringing a new idea or product to market), the view of a security enhancement as innovation is best understood as a negative addition (i.e., removing some bug, vulnerability, or exploitable hole in a product). These security innovations are the same as feature innovations in that they consist of new lines of code and result from the intellectual enterprise of their creators. Nonetheless, while a new feature innovation is seen by a product’s user as an addition, a new security innovation is at best viewed as something that exists in the background and

adds little in terms of functionality and at worst is seen to subtract from functionality for the sake of security. A good example here is encryption, which is a brilliant security innovation that most users ignore unless forced to do otherwise due to the time and effort involved in using it.

This view that security innovations are of mixed value, however, sharply contrasts the view of the software developers and coders, who see security innovations as an indispensable component of the FOSS advantages. Esler, the Sourcefire open source community manager, stated plainly that the idea of open source being more secure was “one of our (Sourcefire’s) fundamental beliefs...and it’s proven it time and time again.” He went on to provide an example in which the Google Project Zero research team – which is considered among the best security teams in the business – identified a serious issue in one of Sourcefire’s products that would allow it to be crashed remotely. In addition to identifying the vulnerability, the Google team also provided a patch to Sourcefire. While the total time between identifying the problem and releasing the patched version of the software was a few days in this case, it could have been even shorter – even just a few hours. Such a turnaround would not be possible in the proprietary software development world due to the greater length of time required before code becomes available to security researchers. For FOSS products, code is essentially available in real time as the software is being developed, whereas the code may not be accessed for non-FOSS products until the owners choose to release it. As Esler noted in relation to the above example, the Google team would not have found – and solved – the vulnerability as quickly in a non-FOSS product.

Other Snort developers, including both Sourcefire employees and FOSS community members, echoed these sentiments. Matt Mickel, an employee tasked with working closely alongside the FOSS community, shared an example of a community contribution: a signature that marked a high-level vulnerability. While the vulnerability was of such importance that Mickel refrained from sharing further details (citing security concerns), the example demonstrates both the importance of security and the role played by the external knowledge of the FOSS community. In the minds of these FOSS developers, security is not a necessary evil but an essential part of the software program they are working on. Improvements and innovations to the program's security are just as important and worthy as changes in features that are more outwardly visible to the user.

The statements by those involved in the Snort project reflect widely held beliefs in the FOSS community – and to a lesser extent the security community – that open source products are more secure as a result of the greater number of people working to find new ways to close security holes in the software (namely the kinds of ‘negative’ innovations discussed previously). Open source pioneer Eric Raymond expresses the faith in the open source community's ability to solve problems as thus: “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone...or, less formally, given enough eyeballs, all bugs are shallow” (Raymond, 1998). Dubbed ‘Linus's law’ after Linux creator Linus Torvald, this idea is at the very foundation of the FOSS movement.

The depth and breadth of the community of contributors hunting bugs in the Snort project confirms that this ‘many eyeballs’ approach can work in a real setting. Of the more than 1500 individuals who joined the Snort open source community by participating

in the email listserv conversation during the study period, 23.91% (or 369) of those individuals reported having found a bug in the program. This is a healthy rate of participation, particularly considering that only 11 of the 369 were Sourcefire employees. Additionally, while those employees identified 97 project bugs during the study period, the non-employees identified 688 bugs. Even if this almost 1:1 ratio were adjusted to account for the fact that a great deal of small bugs and fixes identified by Sourcefire internally were likely fixed without being mentioned on the listserv (due to their minor nature), the volume of security issues being flagged as a result of the FOSS community is still significant enough that it cannot be discounted.

A comparison of the data describing bug contributors from the FOSS community and the data discussed in Chapter Four concerning contributors of innovative new Snort features also reveals a stark contrast. Whereas contributors of new features also occasionally identify bugs, they are much less likely to do so: they submit an average of 2.20 new features vs. an average of only 1.36 bugs. Similarly, bug contributors are far less likely to submit new feature suggestions: they suggest only 1.26 features vs. 2.13 bugs on average. The difference is similar when isolating non-Sourcefire employees (i.e., FOSS external knowledge contributors): here each individual contributes an average of only 1.06 features vs. 1.92 bugs. This data describes separate communities within the Snort FOSS community. One group of security-minded programmers focuses its collective efforts on adding to the relevant body of knowledge in that area, while another, largely separate group of programmers with a clear bias toward adding new functionality is working at simultaneously building that distinct body of knowledge. By fostering both communities, Sourcefire takes advantage of both specialization and a division of labor.

Table 7: Summary statistics for bug contributors

		All Contributors	Bug Contributors Only	Sourcefire Bug Contributors Only	Non-Sourcefire Bug Contributors Only
Messages	Total	8891	5854	1924	3930
	Mean	5.76	15.86	174.91	10.698
	Median	2	3	67	3
Days Active	Total	-	-	-	-
	Mean	174.14	431.02	1963.35	383.94
	Median	2	31.62	2085.88	36.18
Bugs Submitted	Total	785	785	97	688
	Mean	0.51	2.13	8.82	1.92
	Median	0	1	3	1
Code Submitted	Total	1080	907	188	719
	Mean	0.7	2.46	17.09	2.01
	Median	0	0	5	1
Features Submitted	Total	755	465	89	378
	Mean	0.49	1.26	7.91	1.06
	Median	0	1	3	0

Managing the community and ensuring that external knowledge inputs are handled properly can be a critical part of the security/innovation function. A relevant example here is the high-profile Heartbleed bug, which became public in April 2014, and affected the OpenSSL protocol. OpenSSL is a FOSS program that provides cryptography support for web applications, and the Heartbleed bug was a vulnerability that made some data that should have been hidden viewable. At the time, many observers speculated that this bug represented a failure of FOSS – one headline read *Heartbleed: Open Source's Worst Hour* (Vaughn-Nichols, 2014) – and a possible example of the failure of ‘security through transparency.’ FOSS proponents of course argued that FOSS had succeeded, mainly because it was ultimately the community that found and fixed the vulnerability.

Moreover, the evidence in support of this claim is strong. While some malicious actors were able to successfully exploit the Heartbleed bug after it was made public, no cases of the vulnerability being exploited prior to its discovery were confirmed (Gallagher, 2014; Errata, 2014).

In the Heartbleed/OpenSSL case, the need to understand how and why the failure occurred is of great relevance to the open source/security question. In the days and months after Heartbleed's discovery, significant criticism (see for example Roberts, 2015) was leveled at the FOSS community for its failures to catch a significant bug. However, the same critics have also focused on the relative lack of FOSS community participation in OpenSSL development. Indeed, it has been noted that OpenSSL has only two full-time programmers and that contributions from the community are minimal due to a lack of engagement by project managers (Gallagher, 2014). This analysis of the Heartbleed episode matches the more general conclusion offered by Witten et al. (2001): "access to source code lets users improve system security—if they have the capability and resources to do so" (Witten et al., 2001).

In the above example, a program that some estimates claim is used by nearly two thirds of all websites, two individuals were responsible for the bulk of the programming, with little assistance in growing or incorporating the efforts of a greater community. The example of Snort, which is a market segment leader but does not enjoy the same widespread adoption of OpenSSL by any measure, stands in contrast. Even though Snort is lagging in adoption, however, Sourcefire employs a team of dozens of full-time programmers, several of whom dedicate all of their time to working with and managing

the community. This has paid off, as reflected by the high volume and diversity of external knowledge contributed to security concerns related to the program.

In addition, Snort's managers have established a clear, identifiable, and well-known system of governance for the project. This system consists of three major planks: bugs and other security issues, new contributions, and projects (see Chapter Six for a full discussion of Sourcefire's governance strategy for Snort). Joel Esler, explained the bugs plank as follows:

As far as legal is concerned...bugs are 'Hey, we found a problem with your code, we need to fix that problem, here is a patch to fix that problem'. Very simple. It fixes this one issue. And that's kind of what we classify as a bug. And, largely, bugs, what we've come up with is basically a procedure for us to say 'Hey, here's a bug.' And what we've done internally is say 'This is the person who's written the patch and is communicating with the community. This is where the community code is hosted. Does the manager know?' And that's that. Right so, if you worked here (Cisco) and you wanted to go fix something, all you have to do, say you work for me, you let me know and I say 'Ok, that's good' you fill out a form, real small, a couple things, and you're done. And that's the way we're restructuring bugs. Right now, it's a very convoluted, gigantic long document that they have to fill out and it has to be reviewed by lawyers, has to be reviewed by strategists, reviewed by their manager and that kind of thing. So we're streamlining that down.

While revealing that governance is an ever-developing process, these remarks also demonstrate what good governance might look like. First and foremost, a process must be followed, complete with rules and a chain of command (including programmers, managers, and lawyers). Second, communication is emphasized: a procedure for accepting community input is in place, and making managers aware of new bugs is given priority. Third, credit is important. An effort is made to keep a record of who submitted each bug, which is of additional importance given that giving fair credit is a part of the

open source ethos. Finally, a paper trail that can be followed and referred to in the future is maintained.

Snort's FOSS community has an overwhelming belief that training more eyes on a problem will lead to more and better solutions, and that work in pursuit of this goal constitutes a significant part of the FOSS community's contributions to Snort. Good community management ensures that these efforts are realized in the form of a better, more secure, and therefore more innovative product. When it comes to security issues in particular, an organizational structure and culture to go with it attract a specific type of contributor to the community, which in turn expands the depth and breadth of the external knowledge flowing into Snort.

Characterizing Security Contributions

The affiliations of bug contributors also provide additional clues about who adds to the project and why, as well as to how external knowledge finds its way into an open source project via a community of contributors. The affiliations of 117 of the 369 bug contributors were identified using email address domains or signature block information. As may be expected, bug contributors are often associated with security firms (including MITRE Corp., Ball Aerospace, and Northrup Grumman), government and military agencies (e.g., the DoD, the U.S. Department of Energy, and Los Alamos Labs), and other security-minded operations (such as the non-profit Honeynet Project). Tellingly, other than Sourcefire itself the organization with the most contributors (namely five) was a firm called Silicon Defense Inc., a now defunct network security company whose employees went on to join Fireeye Inc. (which is currently considered one of the world's leading network security firms). However, contributors also came from academia (e.g., Duke University, Wayne State University, and Oregon State University), non-security

businesses (e.g., Stamps.com, McGraw-Hill, and Lands' End Inc.), and other sectors (e.g., American Beef Processors of Oregon and the Open Information Security Foundation).²

Affiliations also reveal a distinct international flavor in this particular segment of the Snort community. Belgium, Germany, New Zealand, France, the United Kingdom, Russia, and Spain were all represented in the submissions, in addition to the United States. It is noteworthy, however, that no agency or department of a non-U.S. government was overtly attributed to any community member. All foreign entities were either corporations or individuals (often graduate students or university researchers) who offered contributions as part of private sector or academic interests.

A great deal of information about the contributors as individuals can also be obtained by taking a closer look at the data for the study period. For example, the majority of bug contributors tended to have limited interactions with the Snort community and participated only for the purpose of reporting an identified bug. Indeed, the median number of days that a bug reporter was active on the Snort listserv was only 31.63 and the median number of messages sent was only three. These low numbers are indicative of a fleeting 'fly-by' relationship that is neither strong nor long-standing. The individuals who report bugs may thus have no real commitment to Snort or the community and seem to be acting only to share their own small discovery, not to be a part of some larger whole.

However, the data also shows that, despite relatively low median numbers for days active and messages, the means of those same statistics are much higher. Close

² For a complete list of bug contributors' affiliated firms, see Appendix I.

examination reveals the cause of this difference to be the presence of a small but very active group of contributors who are quite the opposite of the typical ‘fly-by’ bug reporter. This other group of ‘super’ contributors is characterized by a persistent presence both in the community and on the listserv, a much higher level of participation in the discussion, and a demonstrated willingness to not only observe but also to actively participate in the community by submitting bugs.

Table 8: Behaviors of all bug contributors

	Bugs Contributed	Messages	Days Active
Mean	2.13	15.86	431.32
Median	1	3	31.63
Mode	1	1	1

The difference between the fly-by and super contributors can help to build an understanding of who is in the community and why, as well as reveal much about how the community works and what drives innovative activity. For example, combining observations regarding individual contributor behavior and information about contributor affiliations create an opportunity to further understand which groups are driving the most significant activities of the FOSS community and what may be giving rise to the most innovation. For instance, only 56 of the 369 individual bug contributors reported 3 or more total bugs. Despite comprising only 15% of bug contributors, that small group reported 418 (or 53%) of the 785 total bugs submitted.

More interesting is that of the 56 frequent bug reporters, only 3 did not provide code with any of their bug reports; the other 53 provided code at least occasionally. Comparing these rates to infrequent contributors, a sharp contrast stands out: of the 313 individuals who contributed only 1 or 2 bugs, 116 did not submit code (37%). This core group of frequent contributors driving the security aspect of Snort's FOSS activity is thus contributing external knowledge twice (first by identifying bugs and then again by submitting new code to address those issues), which illustrates the importance of the connection between the security issue and innovative activity.

Table 9: Behaviors of fly-by vs. super contributors

	Total Number of Contributors (%)	Total Bugs (%)	Contributors Also Providing Code (%)	Contributors Not Providing Code (%)
3 or More Bugs Contributed	56 (15%)	418 (53%)	53 (95%)	3 (5%)
2 or Fewer Bugs Contributed	313 (85%)	367 (47%)	197 (63%)	116 (37%)

Just as the affiliations of bug contributors helped to form an understanding of who these community members (in contrast to non-contributors), affiliations can also help to understand the differences between super and fly-by contributors. Super contributors' affiliated organizations are much more reflective of a focus on security commensurate with the task of identifying and correcting bugs – the list is nearly completely dominated

by security-minded firms and organizations.³ Gone are firms such as McGraw-Hill and Lands' End, who focus primarily on other commercial ventures. Also gone are the majority of academic researchers, with those remaining clearly being members of the security community (e.g., members of Carnegie Mellon's Computer Emergency Response Team, or CERT). Even the presence of government entities has dwindled to include only two agencies whose missions are obviously in the realm of security, namely the aforementioned CERT and Los Alamos Labs. The remaining organizations are all computer, information technology, or technology firms with a compelling interest in network security.

Security and External Knowledge

The research on motivations for contributing to FOSS communities is vast and shows several possible drivers, including prestige (e.g., Kroah-Hartman, 2009; Bitzer et al., 2007), self-interest in improving a product (e.g., Hertel et al., 2003), and simple pleasure in the task (e.g., Oreg and Nov, 2008). Nonetheless, the case of Snort demonstrates that something else is at work in some cases. The affiliations of super contributors reveal the existence of a community of interest at work within the greater Snort FOSS community. This community is created by the intertwining relationships between the primary governing firm (i.e., Sourcefire), like-minded firms and organizations in the computer security field, and the individuals who work in this world (who often shift between organizations). Such a relationship has important implications for external knowledge and innovation in any open innovation setting.

³ For a complete list of bug super contributors' affiliated firms, see Appendix II.

In Chapter Four, it was noted that one of the primary methods used by Snort and Sourcefire to absorb external knowledge is to hire from the FOSS community, utilizing that network to identify and even develop talent. This intellectual capital and programming talent can move in both directions. Chris Green, who served as Sourcefire's open source community manager for nearly two years, left the firm in late 2003 but continued to be involved well after his departure – even contributing multiple bugs. Joel Esler noted that the knowledge bridge is open in both directions and that talent often moves back and forth between Sourcefire/Cisco and other firms: “We’ve had some people leave, and we’ve had people leave and come back. They call it the ‘grass is greener.’ People think the grass is greener somewhere else so they leave here and realize this is the best job you ever had in our life. And they come back.” Within the community of interest, however, any such movement – regardless of the direction of travel – is far less likely to be completely lost to the Snort project. In addition to Green's continued involvement, FOSS community member Bill Parker noted that he continued contributing to the project even on his own time after completing his normal work at a security research company – despite turning down potential employment opportunities with Cisco (Sourcefire's parent company) to work on open source projects.

Even more than seeing employees and community contributors transition to other roles, the community of interest is about the spirit of sharing that is at the core of the FOSS ethos. When asked about the sharing spirit within Snort's FOSS community, Sourcefire employee Matt Mickel pointed out that the sort of esprit-de-corps fostered by the community of interest is what provides for all of the innovative work that follows,

observing “there’s the aspect of it that’s keeping the open source community alive and thriving. I think that’s its own benefit.”

In both interviews and remarks in the listserv, Snort’s FOSS community members consistently use the phrase “just wanted to do my part” when talking about their contributions to the project. This reflection of a greater sense of community and a fair trade in which individuals receive the source code to an industry-leading product for free in exchange for their best contributions is a core principle of the FOSS movement – and one that appears to be alive and well in the Snort community. This sense of community goes a long way to explain how and why knowledge is shared in that arena. A close relationship that is somewhat symbiotic in nature (in that both parties benefit) binds the external knowledge from a subset of the FOSS community that shares a core goal and belief together with the Sourcefire team, in this case security. If other subsets exist and were to be nurtured by projects leaders, similar benefits would be available to Snort or any other open innovation project.

CHAPTER SIX: GOVERNANCE AND OPEN INNOVATION IN SNORT

Introduction

In the knowledge commons of Ostrom (1995), of which the FOSS community is no doubt a leading example, the community of interested parties is left to manage itself without outside regulation or governance. However, this does not mean that no governance will exist; to the contrary, in the commons a community forms its own rules and order to manage shared resources or knowledge. This is in some ways the fundamental point – a self-governed community is thought to be superior because it allows for a variety of benefits, not least of which is the avoidance of the transaction costs that would be associated with outside regulation (IPRs, in this case). The choice to go FOSS is thus not an abdication of rules and norms, but rather a choice of governance: Who will establish these rules and norms, and how they will go about doing it?

The key question is then whether the choices made by the community on *how* to conduct this governance make any difference. Can a community enhance the positive benefits of the commons by establishing good governance? And does poor governance lead to a diminishment or elimination of the same potential benefits? In reality, governance is a frequently discussed part of open innovation and FOSS. One commentator describes the three ‘pillars’ of FOSS governance as transparency, setting parameters, and making everyone successful (Harvey, 2015). In recent research, Jensen and Scacchi (2010) describe socio-technical interaction networks that help form FOSS governance and according to their findings are important for project success. Lakemond et al. (2016) argue that the governance of management practices is a critical component of successful open innovation in FOSS projects. In the case of Snort, it appears that

governance has played a critical role at every stage of the project's development. In addition to demonstrating that governance matters, this also describes important information about the governance of other FOSS projects.

Models of Governance for Software Innovation and Open Source

The software industry has a robust history of developing software by adhering to a known process of identifying needs, designing and testing solutions, and then implementing these solutions. The process can naturally take many forms, and new ideas about how to do business arise from time to time; however, as taught in computer science and practiced by software firms, all iterations fall under the broad scope of the software development lifecycle (SDLC). In one form or another, the SDLC is widely accepted as the industry standard for new product and process development. McConnell (1996) points to no less than a half dozen different SDLC models that are applied within the software industry; the Waterfall model, which was the first, is the most discussed⁴ – although McConnell also argues that it is perhaps not the best. Larman and Basili (2003) offer incremental and iterative development (IID) as a “modern alternative” to the Waterfall model, even though they assert that IID was originally used in Bell Labs in the 1930s before being applied in the software industry. While IID is a somewhat more deliberate process than most of those described by McConnell, in essence it is simply another way to approach the SDLC.

While these examples illustrate the differences in approach and structure that exist, the SDLC models all have some important similarities. First, they include a few key steps or stages, including planning, developing and building code, testing, and

⁴ See Appendix III for illustrations of SDLC models.

releasing new versions of the software product (see Figure 11). The order, flow, and general work done during each stage vary among the models, but each models includes the same key pieces in some way. Second, each version ends in the release of either a new software product or a new version of an existing product. In the context of understanding how innovation works in the software industry, this fundamental agreement among the SDLC models has clear implications. It demonstrates how knowledge is added to a software product (be it new or existing) and presents a window for viewing how innovation works in the industry.



Figure 11: Stages of the SDLC (BearingPoint, 2012)

With many diverse approaches to the SDLC available, it can be assumed that while each organization may employ some version, the exact impact on the particular organizations' development efforts will depend heavily on the management and governance approach used by the individual group or firm. Whether or not to use FOSS is

only one such management decision, but it is a potentially meaningful one, both for innovation outcomes and for further development.

Governance and the FOSS Approach

Previous research (Grand et al., 2004; Alexy et al., 2013) has shown that software developers can utilize the FOSS community in three ways; they can 1) use existing code by accepting it into their own projects, 2) contribute code to outside projects in a limited role, and 3) make significant contributions to a project by managing it or by initiating a new project. It is already clear that members of the FOSS community have different motivations and goals. The vast majority of Snort's users utilize the community in the first way – they use the open source code that is made freely available. Most of these individuals and firms will use part or all of the code to operate the program, without returning anything to the community. The next largest group chooses the second option: they take time and effort to contribute to the project in some way. This group is unofficially defined as the participants in the listserv, where all contributing activity occurs. For Snort, the final option is selected by the core Sourcefire developers who created and manage the project.

If the goal is to contrast FOSS innovation with closed innovation, those in the first group are of little consequence for innovation – these individuals are users, not contributors. The second group (which is discussed in detail in Chapters Four and Five) is of significant interest for understanding innovation outcomes, as the existence of the FOSS community is the primary differentiator between open and closed firms.

The third group is of interest in comparing how a closed source software firm manages a project vs. how a FOSS firm manages a project *while including the FOSS community*. There may be other differences between the two types of firms (e.g.,

corporate culture) that affect innovation outcomes, but measuring these differences would require a large cross-sectional study and is beyond the scope of this research. For the purpose of understanding how the existence of the FOSS community affects innovation outcomes, the management of the FOSS project by the group or firm that is developing the project is therefore of highest interest.

The addition of the FOSS community necessarily adds a layer of complexity to any application of the SDLC. Instead of only being required to manage developers, information, and knowledge that are directly under their control, the managers of a software development effort must now have a means for addressing knowledge and people external to the normal ecosystem.

According to a 2012 study undertaken by business consulting firm BearingPoint, software firms using FOSS generally break their governance actions down into five categories – one to match each step of the SDLC (see Figure 12). These categories are as follows:

1. Acquisition – the process of identifying the code that can help a given project;
2. Approval – ensuring that any acquired code meets legal, technical, and other standards for inclusion;
3. Cataloging – creating documentation and tracking mechanisms to record and trace the origination of different pieces of code;
4. Validation – making sure that the code in the program work and that no bugs or other errors are present; and
5. Monitoring – ensuring that no future issues arise once the code is deployed (BearingPoint, 2012).

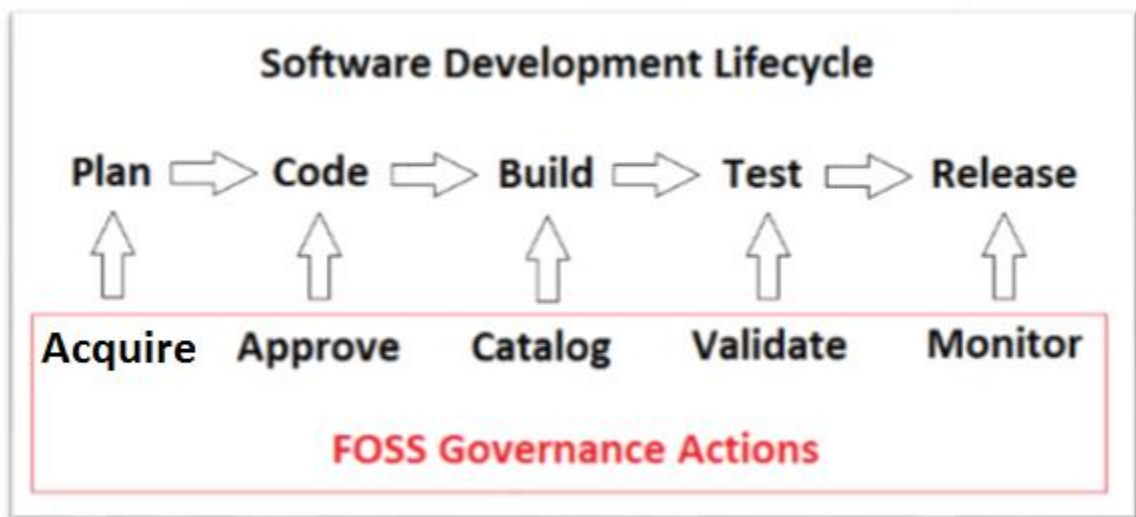


Figure 12: FOSS governance within the SDLC (BearingPoint, 2012)

As the widely accepted industry standard for software development, the SDLC plays a significant role in software innovation. Innovations are a result of many things, including individual effort and ingenuity, collaboration – and sometimes just a bit of luck. There can also be little doubt that processes play a critical role and that the FOSS community can be a component thereof. Nonetheless, an important question still remains: Does this addition increase innovation or simply change the approach to it, making innovation different but no more likely to occur? In other words, how does introducing external knowledge affect the SDLC and any resultant innovation? Answering these questions requires going beyond just describing the governance actions necessary to integrate FOSS into the SDLC to examine how the related changes affect development in a real project, and alter innovation outcomes.

The Snort Project – A Model of Governance in Open Source

The approach used by Sourcefire to integrate FOSS knowledge into the Snort product provides evidence that both the FOSS governance and SLDC models are more than just theoretical. Indeed, significant evidence shows that the way in which Sourcefire manages Snort very closely matches the governance structure shown in Figure 12. However, a close examination of Snort governance also reveals some other notable observations. First, the nature of FOSS development requires project managers to recognize that not all FOSS contributions are the same and hence that they need to use a uniquely appropriate governance structure. Second, the nature of the FOSS community and its relationship with the project and the firm will change over time; governance must be prepared to handle different approaches offered by the FOSS community and shift along with the community.

Snort Project Governance

Sourcefire's approach to managing the Snort project involves segmenting management efforts into three categories of development processes (see Figure 13). The first category, bugs, entails issues or problems with Snort's code that are reported by users in the community. The second category is code contributions, namely additions or improvements to Snort's code that are additive in nature. According to Sourcefire, the last component here is the main differentiator between bug fixes and contributions: a bug fix addresses an existing problem, while a contribution adds new features or functions. The final category is sub-projects, which are large, ongoing development efforts to build complex functions that sit either within or on top of the Snort program (such as plugins and service packages). Sub-projects are differentiated from the other two categories by their ongoing nature and the fact that they may be co-managed by individuals or

organizations not entirely within Sourcefire, as described in interviews with Joel Esler and other Sourcefire personnel.

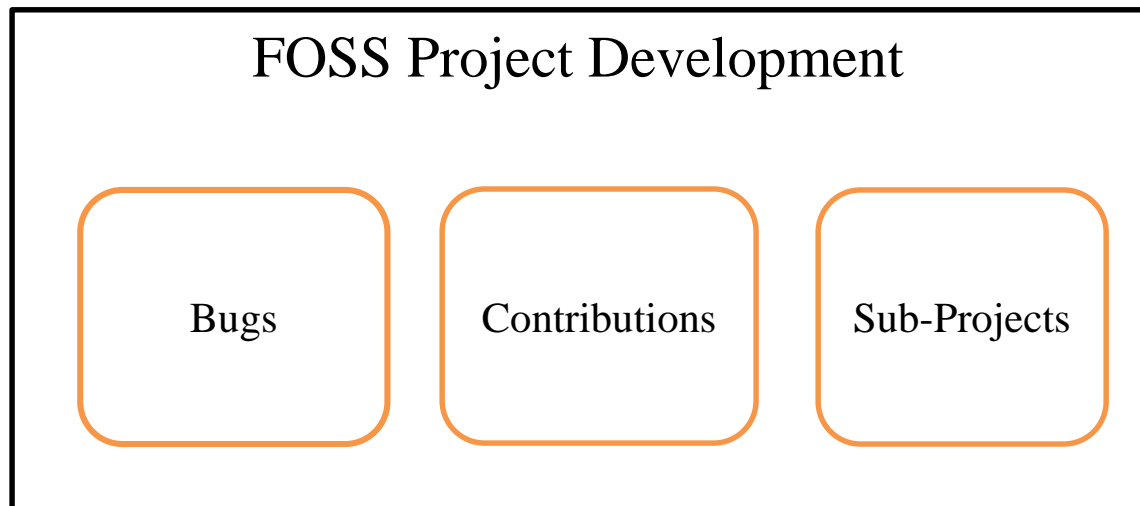


Figure 13: Sourcefire/Cisco open source project governance structure

The result of this three-pronged approach is that Sourcefire can tweak the development cycle and its governance of FOSS input for each category. For example, in managing bugs emphasis is placed on fixing errors rapidly; Esler notes that to accommodate this, Sourcefire has intentionally shortened the approve/catalog/validate steps so that fixes can go out as quickly as bugs are reported (Chapter Five contains a further discussion concerning bugs). On the other hand, contributions go through a more complete vetting process. Because contributions (which are positive additions) most closely resemble the conventional understanding of an ‘innovation,’ they present the best opportunity for understanding FOSS innovation. While the process for managing

contributions is always subject to review, it is this Sourcefire company policy that guides how external knowledge is brought in from the FOSS community.

The above-described governance structure can then be summarized as a process with the following six identifiable steps:

1. A contributor submits a new idea or code via the listserv.
2. A community liaison or other member of development team who is active on the listserv launches a review and coordinates with the rest of Sourcefire's development team.
3. Sourcefire's lawyers and other experts conduct a legal review.
4. Sourcefire programmers bring code up to Sourcefire standards, in terms of both quality and ability to fit into the Snort ecosystem.
5. Programmers then implement the code, adding new features or fixing old mistakes within the program.
6. Sourcefire gives credit to contributors.

This process closely resembles the theoretical model of the SDLC with open source governance as described earlier in this chapter. For example, steps 2 and 3 above closely match the 'approve' and 'catalog' steps of the FOSS governance approach. As a business entity, Sourcefire wants a legal review of contributions (which constitutes an approval) as well as a technical review. Moreover, there can be no doubt that these reviews include some effort to catalog the new additions. Similarly, step 4 matches the earlier described 'validate' component of the SDLC. In this stage, Sourcefire is ensuring that the code fits into Snort and that the program works as desired with the new code. A live, functioning version of the software can only be released after this validation.

According to Sourcefire personnel, the process for managing contributions is complex, geared toward protecting the firm's interests (while crediting creators), and focused on integrating code into the Snort project by standardizing it. As Esler states:

Then you've got what I term as a contribution. This is significantly more than a bug but not an entire project. So this is something like, you want to write a patch into Open SSL to accept this new encryption language. Large change, right? But it has potential ramifications not only on this project but also for Cisco. So, that's a contribution, something much bigger. And that requires a little more paperwork, has to be reviewed by legal, patent attorneys, things like that to make sure all our t's are crossed, our i's are dotted. Make sure you're not giving away intellectual property.

Nonetheless, legal vetting is not the only review that a new contribution must go through in order to be integrated into Snort; ensuring that the code also meets corporate standards for quality and functionality is equally important. As Esler goes on to explain:

All code has to be brought up to our standards. Doesn't matter if it's perfect, it has to fit into our ecosystem. It has to fit our development life-cycle. And then you've got an entire commercial product behind [that] too, you have to remember. So we have to make sure the commercial product works off of the output of whatever new feature you've just built in." On the back end, credit for contributors is also important. The credits and changelogs clearly indicated the names of all those who have contributed code to the project.

In observing the interactions on the listserv, it is clear that the governance structure, as described by Sourcefire personnel, is implemented largely un-changed in the FOSS community. In fact, Sourcefire employees regularly use the listserv to send messages that remind the community of issues such as how the process works and how to share code. For instance, a 2004 message from Sourcefire employee Dan Roelker stated:

Patch submission for Snort is also more rigorous these days. Not only does a patch need to add some sort of enhancement, but it needs to be coded

well and go through a code audit and our testing. It's not a simple matter of running that patch through a test suite to determine it's ok. It requires much more time than that, especially if it's a patch of any significance. And since all the Snort developers are busy adding new features, we add these patches to our queue for when we do have time. We only break that rule if the patches add substantial functionality that we want to add right away.

This process describes in great detail what happens to knowledge once it hits the 'corporate' side of Sourcefire/Cisco – that is, how Sourcefire/Cisco oversees the legal and business structure for building Snort. This part of Snort development is entirely under the control of the firm. However, the process also describes other steps, some critical, during which development and external knowledge are not under the control of the firms (at least not completely). Observing the interactions between the community and Sourcefire/Cisco is a powerful tool for understanding how new knowledge contributions flow from the greater Snort ecosystem into the firm-controlled development process.

An example of an early conversation from April 2001 reveals how Snort's FOSS developers address the cycle of releasing new versions of the product and manage adding new features that are being prepared as a release date nears (see Figure 14). Of note is that the two developers involved in the conversation are both FOSS community members, not core Snort project developers. The nature of the conversation suggests that they nonetheless have considerable input into the process beyond contributing new code. The community members also have an obvious sense of ownership. While a Snort team developer ultimately weighed in and settled this particular case by deferring work on this project until after the release date, the employee committed to making this effort a top priority for the next release. The community's level of involvement demonstrates that

project leaders must take this group into account when establishing governance and management procedures.

Re: [Snort-devel] snort leaders, I need guidance - help!

From: Brian Caswell <bmc@mi...> - 2001-04-23 21:50:32

Todd Lewis wrote:

> Marty, I would like for the paengine stuff to be integrated into 1.8 and
> any plugins that depend on pcap be broken until their authors fix them.
> As you know by my constant annoying messages to the list, I have nothing
> but time to work on this problem and happy to help the plugin authors
> work through these problems. The alternative, of just giving up on
> paengines for 1.8, just means that we're going to have the exact same
> problem the next time we try to integrate it.

This is my take on it. We are -> <- this close to releasing 1.8. Do we want to
massively break a lot of plugins right before a release? If we are going to use
paengines, then we need to do this
AFTER our next release. We should NOT do a release with broken plugins.

Yucky. Todd, if you finish your code to get it READY to integrate, then we can
integrate into CVS as soon as we do a release. We should be focusing on cleaning up
bugs, not massively changing
things.

Should we start a release cycle akin to pick-your-bad? (Every X months, with a lock on
the tree 1 month before hand with only bug fixes, documentation and tested rules added?)

Pretty soon we are going to have ghant charts, UML models, and managers telling us that
we need to use OOP VB. Yucky.

-brian

Figure 14: Snort-devel listserv conversation 6

Another example from the same timeframe shows how the Sourcefire team uses the FOSS community in the beta-release process. In July 2001, Martin Roesch put out a call for the community to review the beta version of Snort 1.8.1:

[Snort-devel] Snort-1.8.1-beta3 uploaded to CVS

From: Martin Roesch <roesch@so...> - 2001-07-20 15:45:26

Ok, it looks like things are finally stable on all reporting platforms. I've just uploaded version 1.8.1-beta3 (build 47) to CVS, I'm going to let people run it for the next several hours and see if any problems crop up, if not we'll wrap it up and call it 1.8.1.

Those of you testing the code, please download it and have a look.
Thanks.

-Marty

--
Martin Roesch
roesch@...
<http://www.sourceforge.com> - <http://www.snort.org>

Figure 15: Snort-devel listserv conversation 7

Snort 1.8.1 was subsequently released in August 2001. In the intervening time, community members made no less than a half dozen suggestions to improve it. While there is no indication that any were significant, the iterative nature of software development means that all suggested changes were nonetheless important parts of Snort's growth. It is also noteworthy that this sort of direct solicitation of feedback and input from the FOSS community is far from the exception; as a general rule, the release of a new version of Snort is always preceded by a beta release that is shared with the community and followed by an announcement to publicize it and solicit feedback.

Perhaps because of this, conversations on the listserv often appear to revolve largely around the release dates of new versions of the software. A significant focus is given to both working to ensure that late additions are finished in time to be included in the release and receiving post-release feedback that can help with future development efforts. While the pre-release efforts (e.g., the addition of new features) are proactive in nature, post-release conversations appear to be more reactive in nature, covering issues

such as efforts to fix bugs or other errors in the program. The data from the listserv archive suggests that the post-release (or reactive) contribution is responsible for a higher share of spikes in activity within the community.

A total of 67 new versions were released during the study period, representing an average of one new version every 73 days. Out of the 67 new releases, 40 preceded month-to-month increases in messaging activity on the listserv while only 32 followed a month-to-month increase (see Figure 16). Furthermore, the magnitude of the increase in the following months was greater than in the preceding months. In the 40 months that saw an increase in activity following a new version release, the monthly message volume jumped by an average of 19.25 messages (or 72.71%) over the previous month. By contrast, the months that followed showed an average jump of only 17.63 messages (or 71.05%) over preceding months. As noted in Chapter Four, the overall contribution rate has declined as the project has matured (see also Figure 16).

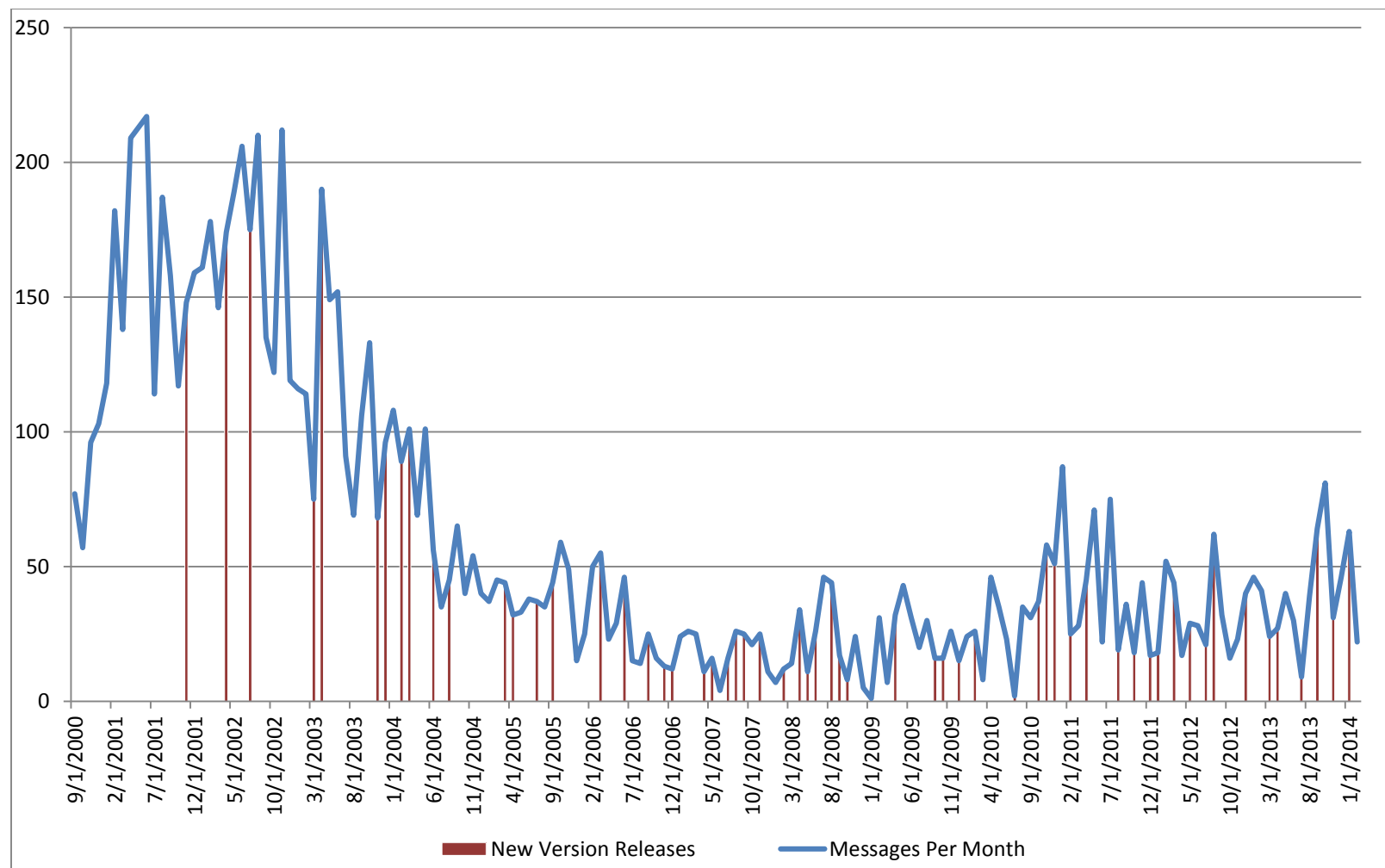


Figure 16: Activity surrounding version release dates

These differences, particularly in the frequency with which new releases are followed by increases in listserv activity, are indicative of an ecosystem in which the project is driving (or pushing) the innovation cycle. In this case, the community is playing more of a reactive role. Recalling FOSS governance and the SDLC, this means that, by and large, when the SDLC ends with a new release and resets to the planning acquisition phase, a signal is sent to the FOSS community that the project is ready to accept new ideas. The new release is studied by community members who identify bugs, generate new ideas, and share their knowledge to start the next cycle.

Changes over Time – the Shifting Sands of FOSS Communities

As with any large, ongoing project, Snort has seen its share of changes over time, and these changes have been reflected in its governance. An example of how Snort and its community have developed is the shift toward greater levels of internal development, i.e., more development being taken on by Sourcefire and the core development group (which necessitated the previously discussed move to secure and prioritize internal development). As noted in Chapters Four and Five, Sourcefire has made significant steps toward building its internal development capacity (which has included – but not been limited to – internalizing community knowledge through hiring as well as by actively seeking community input for internally developed codes and projects). However, as has also been highlighted, rates of contributions from the community (including bug reports and code commits) and participation in the listserv have both declined with time.

Despite this decline in community activity, progress on Snort has not declined. To the contrary, those at Sourcefire report that Snort has actually thrived in recent years,

with the pace of development continuing in an uninhibited manner. This is because the firm's efforts to internalize development have more than offset any loss of community participation. Even though Sourcefire officials such as Esler and Mickel are careful to emphasize that the FOSS community remains as critical as ever to the project's long-term goals and present success, they admit that a shift toward more internal development is indeed taking place. As noted previously, it is estimated that 99% of code is now written in-house by Sourcefire. However, Esler also describes a paradigm in which development has been brought in house to such an extent that source code is now held internally by Sourcefire until a new version is released. Upon release, the community can access this code as always and subsequently provide input, suggestions, and further new code. This approach represents a major shift in governance: the FOSS community now has an opportunity to provide input only on published code. This contrasts with the approach that Snort used previously (and that numerous other FOSS projects continue to employ) of allowing community access to the code throughout the development process.

According Sourcefire employees, this shift occurred for a number of reasons, including the need to better control the company's internal development and improve its quality control and code-writing practices. More importantly, however, they described the process as a slow shift that occurred over a long period of time. According to both internal and external Snort contributors, the project was initially at or near 100% FOSS with no internal component. This initial orientation lasted for some time, but as Sourcefire began to grow, the contribution rate from the community started to steadily decline until things leveled out at the opposite end of the spectrum – near solely internal

contributions. Opinions surrounding the reason for this shift from those in the community differ from the reasons given by Sourcefire personnel (rumors of a desire to protect IP or even end Snort's FOSS orientation are rampant in the community), but broad agreement exists that this shift has occurred.

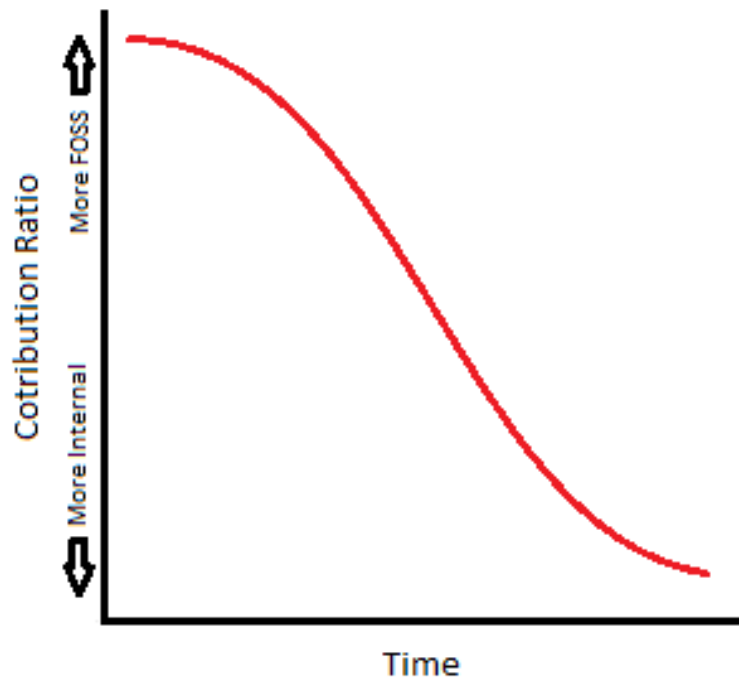


Figure 17: Anticipated shift in contribution types

In addition to the shift toward more internal development and less reliance on the FOSS community for contributions, the FOSS project is undergoing another change as it matures. As described in Chapter Four, Sourcefire has begun hiring its community and

not just passively waiting for contributions to come in. Of course, before Sourcefire was established as a firm, such an approach was impossible. Much as those in- and outside Sourcefire described the gradual move toward internal development, they observed that the move from passively absorbing FOSS community knowledge via the acceptance of contributions to actively absorbing it via direct hiring happened slowly over a period of time. In this case, however, Esler and others at Sourcefire described a more linear decline that developed steadily starting with the founding of Sourcefire, which was originally staffed with community members (see Figure 18). Esler noted that identifying and acquiring talent is now the primary advantage of Sourcefire's access to the community – not the contributions it receives from members. This again denotes a change in the relationship between Snort's controllers and its community: governance is placing more emphasis on bringing talent into Sourcefire than on bringing knowledge directly into the Snort program.

As shifts in the relationship between the community and the core development team are inevitable with the project's maturation and growth, it is equally inevitable that governance will change and adapt to the new reality when these shifts occur. Because the FOSS project both depends on and makes heavy use of the community, one unique challenge in managing the project is that not all parties may agree with the changes that result. By contrast, a closed source project may shift direction or even undergo a complete change in its leaders' vision without running the risk of alienating a significant part of its development community. Snort is no exception when it comes to the possibility of divergent opinions in the FOSS community, as the history of the product's

development shows. This history also demonstrates the critical role that governance plays in how the community reacts to any changes and the impacts these reactions then have on the project.

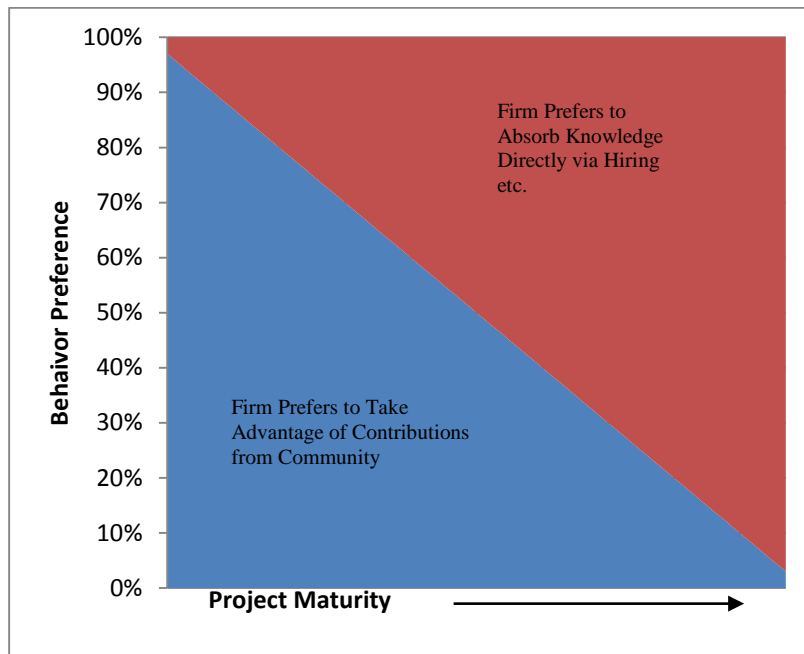


Figure 18: Changes in firm preference toward community

In the case of Snort, one of the most significant consequences of the project's growth – and the concurrent growth of Sourcefire as a commercial company – was friction that developed between FOSS community members (whose primary interest is in keeping Snort as open as possible) and the Sourcefire-led core development team (whose primary interest is operating an open source project and a profitable company). In the words of Chris Green, an early community contributor to Snort who went on to become Sourcefire's fifth employee and a major Snort developer, this led to stress both in the community and at Sourcefire. According to Green, it was "very hard to go from open

source full collaborative to ‘we need to deliver a product that works to paying customers.’ ” One of the major impetuses for the firm’s focus on commercialization was that in the early years of the project, other companies were actively shipping Snort, sometimes with their own enhancements. These competing pressures are what eventually led Chris Green to leave Sourcefire, although he has remained a Snort contributor.

These same pressures also gave rise to concerns within the community that Snort was losing some of its ‘openness’ and that the community’s interests were being somehow marginalized. The first indication of this concern manifested itself in the form of a lengthy listserv conversation that started in July 2002. Jed Pickel, a longtime Snort contributor, began by listing several grievances, mostly related to how code was being integrated into Snort. He ended by stating:

Now is the time for the community to begin discussing forming a branch of Snort that is governed by a consortium that is not profit driven, but rather exists to support the best interests of the community and support healthy competition among all of the companies that are providing Snort based security solutions.

No evidence indicates that any changes were undergone at this point in Snort’s governance, and a new branch of Snort (commonly referred to as a ‘fork’ within the FOSS industry) was not ultimately formed. However, the underlying friction between the community and the core developers at Sourcefire remained.

Some years later, namely in 2009, some members of the Snort community did end up breaking away to move in a completely different direction. According to the developers who were involved in that decision, the impetus for the change was the same

as the one that had created Jed Pickel's frustration seven years earlier. For example, Jason Ish, a software developer who has been involved in Snort since 2009, remarked "as Sourcefire matured I believe they tightened up on contributions from the outside. In fact, I contributed a patch which was rewritten before being committed, and I believe that was done so Sourcefire could limit the amount of code with 3rd party copyright on it." Re-writing the patch would allow Sourcefire to make that part of the code proprietary and not subject to any licensing restrictions put on it by the contributor. Breno Silva, another former Snort contributor, similarly remarked "one of the main reasons to stop contributing to Snort was rumors in the community that people from Snort's core team fail to give credits for people who contribute with code."

The frustration of Ish, Silva, and others eventually grew to the point that they concluded Snort could no longer serve their needs. Each cited these frustrations as motivation for becoming among the first generations of developers of a new FOSS intrusion detection system (IDS) called Suricata. This new program, which competes with Snort, is governed by the Open Information Security Foundation (OISF). This non-profit foundation is intended to be a governing consortium of sorts, much like what was alluded to in a 2002 conversation on the Snort listserv. Former Sourcefire official Chris Green has noted that the OSIF board consists largely of former Sourcefire employees. Indeed, Matt Jonkman, who is one of OSIF's two founders, was a Snort contributor and listserv participant.

Whether this fracturing of the Snort community was an inevitable result of Snort's growth or a preventable occurrence made possible only by Sourcefire's governance

approach is not clear. However, the statements of those involved in the process strongly suggest that governance played at least some significant role. Sourcefire's governance decisions can therefore be linked to a clear drain of talent from the FOSS community supporting the project. The emphasis on protecting Sourcefire's intellectual property (to the detriment of including some community contributions) is perhaps the most obvious and significant example of this. Other examples do exist, however, such as the documented fear of community members that their contributions will not remain FOSS as a result of Snort's governance team seeking to maximize the product's commercial viability. Moreover, while Snort continued to survive – or even thrive – as the above happened, both governance and understanding changes in the community and the project are nonetheless certainly critical to the success of any FOSS project.

CHAPTER SEVEN: CONCLUSIONS AND IMPLICATIONS FOR POLICY

Is Open Source a Path to Greater Innovation?

The main purpose of this dissertation was to explore how external knowledge from an open source community affects innovation in the software industry. This study was meant as a way to explore issues regarding open innovation, the knowledge commons, and how they each affect innovation outcomes. By looking closely at the FOSS/innovation dynamic, this study provides empirical evidence related to a commonly held assumption within the software community – namely that FOSS is a superior approach to software development because it trains many eyes on a single problem. This ‘many eyes’ theory provides the basis of support for the FOSS community, although it also bears striking resemblances to the underlying theoretical assumptions regarding open innovation and the use of external knowledge (via the knowledge commons). As the FOSS community is a useful example of a functioning knowledge commons, so too is the FOSS development process a relevant example of open innovation. However, the literature regarding these issues lacks empirical evidence to support these claims. More specifically, little evidence is available to demonstrate that FOSS is more innovative than software developed under proprietary systems. While this dissertation cannot completely fill that gap, it hopes to contribute to the literature by providing evidence of how FOSS systems gain access to and utilize a vast cache of knowledge from the community – a cache which, because it is unavailable in non-FOSS systems, provides the FOSS development team with advantages when it comes to pursuing new innovation.

This dissertation's analysis is presented in three sections. The first demonstrates both the method by which external knowledge flows into a FOSS development system and the volume of such information that makes its way into the system. It does so by examining the contributions of FOSS community members to the Snort software project, a sector-leading product developed in the open source environment by Sourcefire. Chapter Four demonstrates that, measured by volume, the contributions of the community are diverse and significant and in many ways outpace even contributions to the project from Sourcefire's in-house developers. Snort users, developers employed by other software firms, academic researchers, and other interested parties from an incredibly diverse set of backgrounds have become part of both the community and the project. The implications are clear: there is no evidence or example of such a community giving such a volume of information freely to proprietary software development. The FOSS community thus presents an information advantage that can be used for open innovation.

Part two of the analysis draws on an unexpected discovery from the results of the case study. The perspectives of Snort developers within both Sourcefire itself and the FOSS community reveal that innovation discussions concurrently focus on security (namely de-bugging the program) and preventing future security threats. These developers appear to view debugging and maintaining a secure product as innovative activity. An innovation is normally seen as a new addition or improvement. Security fixes do not fit that definition; in many ways they are quite the opposite. Nonetheless, software developers are quick to note that they consider security equally important to producing

innovative, effective new software. In the case of what they view as ‘security innovations,’ a clear difference exists between the FOSS system and the proprietary approach. Whereas the FOSS community believes that the many eyes approach leads to better security by allowing significantly more experts to view code, keeping source code secret is an integral part of the security regime for proprietary software. This is best understood as security via transparency vs. security via openness. Chapter Five clearly demonstrates that the bulk of bug fixes and related measures come from the FOSS community and that even Sourcefire considers these contributions to be essential in making Snort one of the most secure products on the market. While represents is a different type of knowledge, it is another clear example of how the FOSS approach provides an avenue to greater innovation (via external knowledge) that is not available to other methods of software development.

The final part of this analysis constitutes a review of FOSS governance, based on the approach used by Snort’s development group. As the external knowledge emanating from the commons (i.e., the Snort FOSS community) is a substantial addition to the body of knowledge being used to develop Snort and this knowledge represents a significant departure from the normal development cycle used by software firms and groups, the development cycle must be altered substantially to accommodate the new knowledge. In addition, because the FOSS community may shift over time in ways that the core development group cannot always control, any approach to governance must adapt alongside these shifts. As described in Chapter Six, Snort’s experience demonstrates that governance is integral to both the growth of the FOSS community and the cooperation

between that community and the program's core development group. Snort's successful use of external knowledge from the FOSS community was made possible only because there was a solid plan for moving that knowledge into the system, maintaining a robust community of contributors, and adapting to changes over time. Governance is so critical that even in an established community such as Snort's, disagreements resulting from governance decisions can lead to serious fissures in the community, to the detriment of the project.

Taken together, the three components of this analysis constitute a framework for understanding how external knowledge makes its way into an open innovation system. To be successful, a system must build a large network or community that is capable of accessing a significant body of knowledge. Furthermore, efficacy can be enhanced if the approach allows access to potential forms of innovation that are not available in more traditional models of innovation. Finally, governance of the FOSS knowledge and community is critical; without good governance, the potential benefits of open innovation *may* be lost.

It is the final point from the above framework that is perhaps the most important for addressing the question: is FOSS more innovative than the proprietary approach? The answer appears to be: FOSS *can* provide a volume of external knowledge that innovation systems *may* use to achieve better innovation outcomes. However, it is beyond the scope of this study to conclude that better innovation *will* result. The use of the FOSS approach certainly provides access to a large body of external knowledge, as evident in the vast network of Snort contributors and their additions to the program. Moreover, due to its

very nature the more open approach can – in some cases – result in innovation opportunities that are not even accessible under other models, as in the examples of innovative security fixes and patches being made available to Snort. Nonetheless, the declines over time in both community participation and FOSS contributions to the project are sizable enough that they cannot be ignored. When combined with the testimony of community members (e.g., the comment that Suricata, the competitor product forked from Snort, is “like Snort, only more open”), these declines depict a FOSS commons that is not as open as it once was. As demonstrated by part three of this study’s analysis, governance is critical for seeing external knowledge through to becoming new innovation and – along with other factors – ultimately determines the net benefit of the open approach.

When he speaks of ‘democratizing innovation,’ Eric von Hippel (2005) is referring to users’ ability to innovate within a product. To make this happen, however, users in the community must be able to associate, cooperate, and organize freely. Ostrom’s (1990) concept of a commons is similar: an innovative community of freely associating members who are allowed to manage resources and solve problems in a collaborative way through voluntary organization. In each case, the benefits of these arrangements are not guaranteed – benefits can be threatened by a number of challenges, including the excessive use of IP protections or coercive action. This dissertation demonstrates that the FOSS community, as represented by the Snort project, is an example of how an innovative community can thrive. However, it has also shown that

governance of both the community and the project is a critical determinant of the community's utility.

Much of the information presented above – including that Snort evolved over time to be 'less open' and governance is an important factor – suggests that, at least in later years within the study period, something related to project management was lacking or even a threat to the project's innovative potential. However, no directly apparent link exists between failures in Snort's governance and any changes in innovation. To be sure, as the project grew and evolved, so did the governance structure employed by Roesch and his team at Sourcefire. Moreover, these changes were accompanied by dissention within the community that even rose to the point of many members being disillusioned – as illustrated by their defections to competing projects. Nevertheless, Snort's continued success and recognition as a segment leader (it is now the most widely adopted program of its type) and Sourcefire's ongoing growth together bring into question any suggestion that Snort has stagnated as a product; it is just as possible that these changes are part of a natural evolution as the project grows and expands. To better understand this situation, a long-term comparative case study of maturing FOSS projects could explore relevant issues. However, this is beyond the scope of this dissertation and thus an area for future study.

Policy Implications

Several policy recommendations can be readily deduced based on the research presented above. They are meant to both inform the continued understanding of the

impact of open innovation in general and FOSS in particular, as well as to describe ways in which economies can more easily capture the benefits of open innovation.

Consider New Additions to the System of Patent and Copyright

To be effective, the FOSS commons must be able to exist with clear rules for association that make it possible for community members to organize and cooperate among themselves. Part of this structure involves the need for clear rules concerning how the community treats knowledge contributions. The current IP protection regime is limited in that it does not explicitly include the types of ‘copyleft’ protections that are favored by the FOSS community and other open innovators. A re-evaluation may therefore not have to incorporate any changes to patents, copyrights, or other relevant laws in order to be effective; it may instead include some codification of the open alternatives to these existing protections. This is because the sorts of licenses that constitute the copyleft regime exist only as contracts between the producer and user of the common good or knowledge – which are often just implied or assumed, rather than explicit. For example, software code writers may publish their work under a version of the GPL (a popular FOSS license); in theory, attaching this restriction to a piece of code defines how derivative users of that code may utilize it and spells out requirements for attribution, use for profit, and the licensing of derivative works.

However, no legal document is signed in this exchange. By using code published under the GPL, the subsequent user is instead *implicitly assumed* to have agreed to the GPL’s conditions. The implications if he or she does not adhere to the requirements of the GPL are not clear but may involve elements of contract or IP law and other legal

issues that have not yet been resolved in a general fashion. Some work has been done on the legal implications of ‘copyleft’ licenses (e.g., de Laat, 2005; Brown, 2010), but until significant test cases have worked their way through the court system and resolved just how important these licenses are, the future utility and reliability of these licenses is uncertain. If these licenses prove unenforceable, the structure and therefore the utility of the FOSS community would be threatened. The ‘copyleft’ licenses may thus need to be codified into the IP protection regime in the future if they are to have the legal heft necessary to be effective.

Encourage Investment and Participation in Open Systems

In recent years, local, state, and federal governments have significantly shifted their attitudes on adopting FOSS for use in their computer systems and networks. This should be viewed as a welcome change by FOSS advocates, but further steps could be taken to enhance the utility of FOSS and other open innovation systems. The Snort community includes a significant number of employees from dozens of government agencies. However, it can be expected that most of these entities do not have updated or cohesive policies on either the use of FOSS or employees’ contributions to FOSS projects (if they have policies at all). Clear guidelines on such activities could be made part of the information and communication technology policy of every agency that may have a need for such a policy. Doing so may make government entities and their employees more inclined to join FOSS communities, which would add to those communities’ external knowledge stores and increase their innovative potential. At the same time, these entities would reap the benefits of using FOSS.

Setting Conditions for Communities to Grow and Thrive

Firms interested in profiting from greater innovation, governments interested in seeing greater economic growth, and policy makers interested in setting good policy may not be faulted for simply having set the encouragement of greater innovation as a goal. However, this study and prior research have shown that the best determinants of successful open innovation are good governance and the community's size and diversity. As such, parties interested in encouraging further open innovation in both FOSS and other open spaces would be wise to take both issues into consideration – and not simply settle for open innovation without considering the specifics of the approach.

Recommendations for Future Research

Perhaps the most significant finding of this study is the critical role that the governance of both knowledge and the commons has in realizing the promise of FOSS-driven innovation. Nonetheless, several interesting related questions remain. Will poor governance eradicate the benefits of open innovation completely, or will it simply reduce them? What is the scale of the impact of governance? Which specific parts of governance and knowledge management are most important, and which have no impact? This study suggests answers to some of these questions (for example, regulating how contributors' rights to their contributions are treated and maintaining a sense of commitment to openness seem to be key to good governance), but future research could obtain fuller understandings.

Furthermore, this study examines how the addition of FOSS external knowledge adds to the innovative capacity of a software development team or firm; it does not

address how FOSS knowledge may ‘crowd out’ or otherwise reduce other innovative inputs. For example, does reliance on the FOSS community reduce incentives for firms to hire new developers and thus lower the contributions of internal knowledge to innovation? If this were true, FOSS contributions would simply be substituting (and not adding to) other innovative inputs. That does not appear to be the case with Snort, a recognized leader in innovation – but this question is worthy of further scrutiny.

Finally, this study examined a single case that is, by all accounts, a FOSS success story. Snort’s core development team at Sourcefire has been able to build a large, diverse open community of interested contributors. Using the external knowledge from this community, it has turned Snort into an industry-leading product that is known for innovation and efficacy. A case in which the FOSS approach was not successful – due to failures in community building or implementing community knowledge – would present an interesting contrast. Valuable lessons could be learned by comparing the successful approach of Sourcefire and the Snort community to a case in which less innovation was realized or the product ended up being less highly regarded than its proprietary counterparts.

The dataset compiled for this study provides unique insight into not only the FOSS world, but also to software development in general. The ability to track the collaboration and conversations of those writing code as both ideas that become part of the software and information that is shared and exchanged can describe far more than the role of the FOSS commons and external knowledge. This alone presents an opportunity for future research – but if a database such as the one used in this study could be

compared with a similar database constructed for proprietary software development, the opportunities for new insight would be remarkable. Unfortunately, obtaining such a database is a challenge, which also limited the scope of this dissertation. Proprietary development tends by nature to be protected by its owners, which limits the availability of information. Even if a firm were willing to provide access, the information obtained would likely not be comparable to information from a FOSS setting, which tends to be more distributed and leads to communication that can be easily tracked and coded (as done in this research). Proprietary development may not have the same kind of paper trail to follow.

The main goal of this dissertation was to understand if and how open innovation presents a possible path to greater innovation outcomes than the non-open alternative. The results demonstrate that under the right circumstances, open innovation indeed provides innovators with access to a much greater body of external knowledge that they can use to create innovation. Taken in concert with other research that corroborates these findings, the results suggest that the question posed should shift toward understanding what it takes for open communities to grow and thrive as well as for firms and other organizations to use external knowledge more effectively for innovation.

APPENDIX I – BUG CONTRIBUTOR AFFILIATIONS

A. J. Lill Consultants	McGraw-Hill
Above Sécurité Inc.	Nepenthes Development Team
American Beef Processors of Oregon, LLC	NNSA Information Assurance Response Center (IARC)
Applied Watch Technologies, LLC	NOAA IT Security Office
Arena Consulting Limited	Northern Illinois University
Async Open Source	Northrop Grumman Information Technology
Ball Aerospace & Technologies Corp.	Open Information Security Foundation (OISF)
BBN Technologies	Open-Systems Group Inc.
Berbee	Oregon State University
Bleeding Edge Threats	Paul Scherrer Institut
Bristol-Myers Squibb Pharmaceutical Research Institute	Proseq AS
Buchanan Associates	Provident Analysis Corporation
Cavium IDC	qDefense
CERT/CMU	Riverstone Networks
CleanCommunications	Riverstone Networks, UK
CodeCraftConsultants.com	Sasquatch Computer
Compaq	ScanNet Group A/S ScanNet
Computer Database and Web Solutions Pty Ltd	Seaway Networks Corporation
Consultor TI - ENEO Tecnologia SL	SECNAP Network Security, LLC

Cornell University	SecureCiRT (A SBU of Z-Vance Pte Ltd)
Counterpane Internet Security	SecurePipe, Inc.
Crusoe Researches	Secureworks
D.O.M. Datenverarbeitung GmbH	Securityflaw
D.S.D. Data Security Division	Sensory networks
Data Access Experts	Sentor AB
Data Nerds LLC	Servervault Inc
Diplom-Informatiker	Silicon Defense Inc
Duke University	SourceFire
Easysoft Ltd, UK	Stamps.com, Inc.
Endace	StillSecure
Florida Datamation, Inc.	Symantec, Inc.
Foot Clan	TEAMLOG Nantes
Fox-IT	TELINTRANS
Freie Universitaet Berlin	the Honeynet Project
Gentoo/MIPS	The MITRE Corporation
Gesellschaft fuer Netzwerk und Unix Administration mbH	The University of Auckland, New Zealand
Google Inc.	Thomas Jefferson National Accelerator Facility, US DOE
Graduate Student, Computer Science, University of Maryland	TMP Consultoria em Informatica S/C
Guardent, Inc.	TransComp Systems, Inc.
Humboldt-University of Berlin ZE Computer- und Medienservice	U of MN - OIT

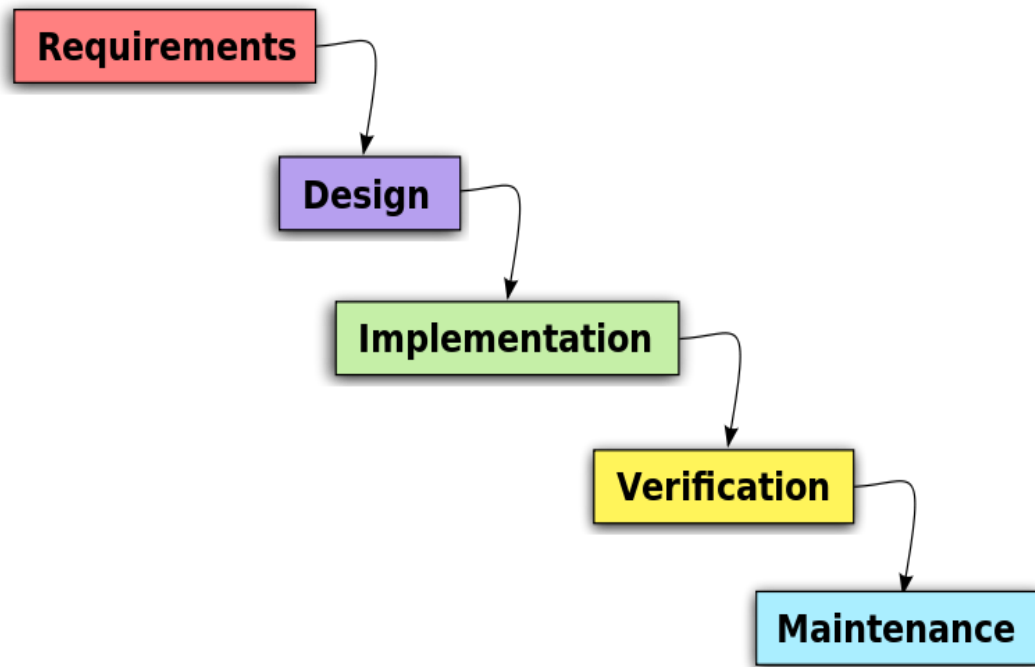
Idaho National Engineering and Environmental Lab. (INEEL)	U.S. Army Research Lab
Imperial College	Universitaet Ulm, SAI
Indiana State University	University at Buffalo
Internet for Learning - AS5503, Research Machines plc. UK	University of Wisconsin Madison, Computer Science Dept.
Internet Operations Center	URIZEN - Internetworking & Digital Security
Internet Partners	Vanderbilt University Medical Center
Inverse Path Ltd	VigilantMinds Inc.
JSC Avtocard-Holding, Moscow	Vrije Universiteit Brussel
Kentucky Department of Education	Wanadoo Belgium NV/SA
Lands End, Inc.	Wayne State University
Latis Networks, Inc.	Webii, Inc.
Los Alamos National Lab	Wurldtech Security Technologies Inc.

APPENDIX II – SECURITY ‘SUPER CONTRIBUTOR’ AFFILIATIONS

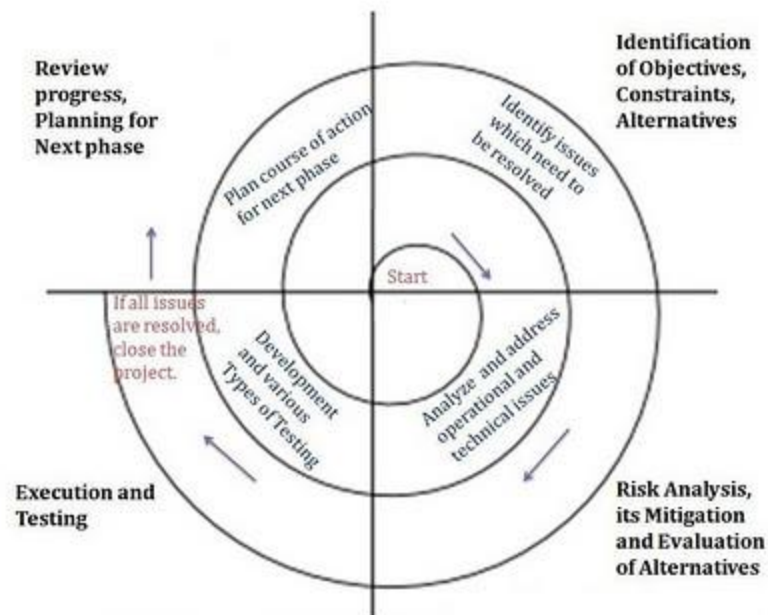
Above Sécurité Inc.
Ball Aerospace & Technologies Corp.
Buchanan Associates
CERT/CMU
Computer Database and Web Solutions Pty Ltd
Crusoe Researches
Gentoo/MIPS
Gesellschaft fuer Netzwerk und Unix Administration mbH
Internet Operations Center
Los Alamos National Lab
NOAA IT Security Office
Provident Analysis Corporation
SecurePipe, Inc.
Sensory networks
Sentor AB
Silicon Defense Inc
SourceFire
StillSecure
The MITRE Corporation

The University of Auckland, New Zealand

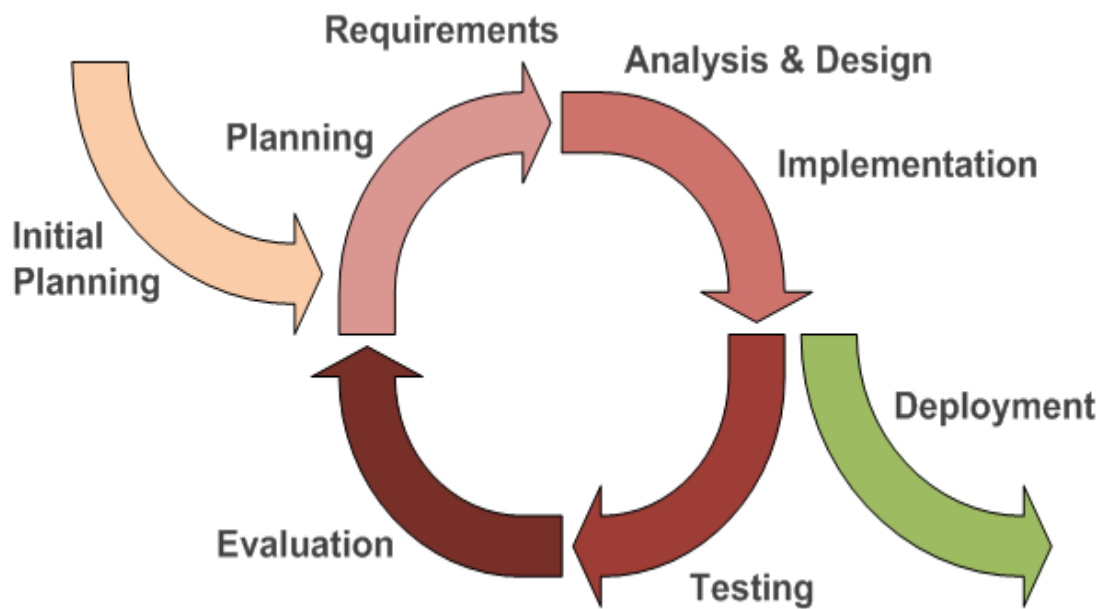
APPENDIX III – SDLC MODELS



The Waterfall Model (McConnell, 1996)



The Spiral Model (McConnell, 1996)



The IID Model (Agile Development, 2015)

REFERENCES

- Aghion, Philippe, Nick Bloom, Richard Blundell, Rachel Griffith, and Peter Howitt. "Competition and Innovation: An Inverted-U Relationship." *The Quarterly Journal of Economics* 120, no. 2 (May 1, 2005): 701–28.
- Alexy, Oliver, Joachim Henkel, and Martin W. Wallin. "From Closed to Open: Job Role Changes, Individual Predispositions, and the Adoption of Commercial Open Source Software Development." *Research Policy* 42, no. 8 (September 2013): 1325–40.
- Amin, Ash, and Joanne Roberts. "Knowing in Action: Beyond Communities of Practice." *Research Policy* 37, no. 2 (March 2008): 353–69.
- Audretsch, David B. *Innovation and Industry Evolution*. MIT Press, 1995.
- Baldwin, Carliss, and Eric von Hippel. "Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation." *Organization Science* 22, no. 6 (November 1, 2011): 1399–1417.
- Bessen, James, and Robert M. Hunt. "An Empirical Look at Software Patents." *Journal of Economics & Management Strategy* 16, no. 1 (2007): 157–89.
- Bessen, James, and Eric Maskin. "Sequential Innovation, Patents, and Imitation." *The RAND Journal of Economics* 40, no. 4 (December 1, 2009): 611–35.
- Bitzer, Jürgen, Wolfram Schrettl, and Philipp J. H. Schröder. "Intrinsic Motivation in Open Source Software Development." *Journal of Comparative Economics* 35, no. 1 (March 2007): 160–69.
- Boudreau, Kevin J. "Let a Thousand Flowers Bloom? An Early Look at Large Numbers of Software App Developers and Patterns of Innovation." *Organization Science* 23, no. 5 (September 1, 2012): 1409–27.
- Brown, Christopher S. "Copyleft, the Disguised Copyright: Why Legislative Copyright Reform Is Superior to Copyleft Licenses." *UMKC Law Review* 78, no. 3 (April 15, 2010): 749–83.
- Bush, George. "National Security Presidential Directive 54: Cybersecurity Policy," January 2008. <https://fas.org/irp/offdocs/nspd/nspd-54.pdf>.
- Chesbrough, Henry William. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business Press, 2003.

- Chesbrough, Henry William, Wim Vanhaverbeke, and Joel West. *Open Innovation: Researching a New Paradigm*. Oxford: Oxford University Press, 2006.
- Coase, R. H. "The Problem of Social Cost." *Journal of Law and Economics* 3 (October 1, 1960): 1–44.
- Cohen, Wesley M., Richard R. Nelson, and John P. Walsh. "Protecting Their Intellectual Assets: Appropriability Conditions and Why U.S. Manufacturing Firms Patent (or Not)." Working Paper. National Bureau of Economic Research, February 2000. <http://www.nber.org/papers/w7552>.
- Crepon, Bruno, Emmanuel Duguet, and Jacques Mairesse. "Research, Innovation, and Productivity: An Econometric Analysis at the Firm Level." Working Paper. National Bureau of Economic Research, August 1998. <http://www.nber.org/papers/w6696>.
- Dahlander, Linus, and Lars Frederiksen. "The Core and Cosmopolitans: A Relational View of Innovation in User Communities." *Organization Science* 23, no. 4 (July 1, 2012): 988–1007.
- Dahlander, Linus, Lars Frederiksen, and Francesco Rullani. "Online Communities and Open Innovation." *Industry and Innovation* 15, no. 2 (April 1, 2008): 115–23.
- Dahlander, Linus, and Mats Magnusson. "How Do Firms Make Use of Open Source Communities?" *Long Range Planning* 41, no. 6 (December 2008): 629–49.
- Dahlander, Linus, and Mats G. Magnusson. "Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms." *Research Policy* 34, no. 4 (May 2005): 481–93.
- Dahlander, Linus, and Martin W. Wallin. "A Man on the inside: Unlocking Communities as Complementary Assets." *Research Policy* 35, no. 8 (October 2006): 1243–59.
- de Laat, Paul B. "Copyright or Copyleft?: An Analysis of Property Regimes for Software Development." *Research Policy* 34, no. 10 (December 2005): 1511–32.
- Demsetz, Harold. "Toward a Theory of Property Rights." *The American Economic Review* 57, no. 2 (May 1967): 347–59.
- Dosi, G., L. Marengo, and C. Pasquali. "How Much Should Society Fuel the Greed of Innovators?: On the Relations between Appropriability, Opportunities and Rates of Innovation." *Research Policy* 35, no. 8 (October 2006): 1110–21.

Edison, Henry, Nauman bin Ali, and Richard Torkar. "Towards Innovation Measurement in the Software Industry." *Journal of Systems and Software* 86, no. 5 (May 2013): 1390–1407.

Esler, Joel. Interview. In Person, December 15, 2015.

Feldman, Maryann, and David Audretsch. "Innovation in Cities: Science-Based Diversity, Specialization and Localized Competition." *European Economic Review* 43 (1999): 409–29.

"FOSS Management Study: Open Source Software." BearingPoint GmbH, 2012. <https://www.blackducksoftware.com/files/survey/fossautostudy.pdf>.

Fruchterman, Jim. "Is Your Open Source Security Software Less Secure?" *Opensource.com*, 2015. <https://opensource.com/business/15/5/why-open-source-means-stronger-security>.

Fulton, Russell. Interview. Telephone, March 11, 2016.

Galinkin, Erick. Interview. In Person, December 15, 2015.

Gallagher, Sean. "Heartbleed Vulnerability May Have Been Exploited Months before Patch [Updated]." *Ars Technica*, April 9, 2014. <http://arstechnica.com/security/2014/04/heartbleed-vulnerability-may-have-been-exploited-months-before-patch/>.

Gates, Bill. "Bill Gates Letter to Hobbyists," February 3, 1976. https://upload.wikimedia.org/wikipedia/commons/1/14/Bill_Gates_Letter_to_Hobbyists.jpg.

Graham, Robert. "Errata Security: 300k Vulnerable to Heartbleed Two Months Later." Accessed April 3, 2016. <http://blog.erratasec.com/2014/06/300k-vulnerable-to-heartbleed-two.html#.U6bXBWSSwyC>.

Grand, Simon, Georg von Krogh, Dorothy Leonard, and Walter Swap. "Resource Allocation beyond Firm Boundaries: A Multi-Level Model for Open Source Innovation." *Long Range Planning, Boundaries and Innovation*, 37, no. 6 (December 2004): 591–610.

Green, Chris. Interview. Telephone, March 11, 2016.

Gwet, Kilem Li. *Handbook of Inter-Rater Reliability* (Second Edition. Advanced Analytics Press, 2010).

- Hall, Bronwyn H., and Josh Lerner. "The Financing of R&D and Innovation." Working Paper. National Bureau of Economic Research, September 2009. <http://www.nber.org/papers/w15325>.
- Hallgren, Kevin A. "Computing Inter-Rater Reliability for Observational Data: An Overview and Tutorial." *Tutorials in Quantitative Methods for Psychology* 8, no. 1 (2012): 23–34.
- Hansen, Marit, Kristian Köhntopp, and Andreas Pfitzmann. "The Open Source Approach — Opportunities and Limitations with Respect to Security and Privacy*." *Computers & Security* 21, no. 5 (October 1, 2002): 461–71.
- Hardin, Garrett. "The Tragedy of the Commons." *Science* 162, no. 3859 (December 13, 1968): 1243–48.
- Harvey, Nathan. "Three Pillars Of Open Source Governance." *InformationWeek*, January 13, 2015. <http://www.informationweek.com/strategic-cio/it-strategy/three-pillars-of-open-source-governance/a/d-id/1318585>.
- Heller, Michael A. "The Tragedy of the Anticommons: Property in the Transition from Marx to Markets." *Harvard Law Review* 111, no. 3 (January 1998): 621–88.
- Heller, Michael A., and Rebecca S. Eisenberg. "Can Patents Deter Innovation? The Anticommons in Biomedical Research." *Science* 280, no. 5364 (May 1, 1998): 698–701.
- Henkel, Joachim. "Selective Revealing in Open Innovation Processes: The Case of Embedded Linux." *Research Policy* 35, no. 7 (September 2006): 953–69.
- Hertel, Guido, Sven Niedner, and Stefanie Herrmann. "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel." *Research Policy, Open Source Software Development*, 32, no. 7 (July 2003): 1159–77.
- Hess, Charlotte, and Elinor Ostrom. *Understanding Knowledge as a Commons: From Theory to Practice*. Cambridge, Mass.: MIT Press, 2007.
- Hoepman, Jaap-Henk, and Bart Jacobs. "Increased Security Through Open Source." *Commun. ACM* 50, no. 1 (January 2007): 79–83.
- "Innovation." *Merriam-Webster*, 2016. <http://www.merriam-webster.com/dictionary/innovation>.
- Ish, Jason. Interview. Telephone, March 8, 2016.

- Jaffe, Adam B. "Analysis of Public Research, Industrial R&D, and Commercial Innovation." In *The Science of Science Policy: A Handbook*, 193–207. Stanford University Press, 2011.
- . "Real Effects of Academic Research." *The American Economic Review* 79, no. 5 (December 1, 1989): 957–70.
- Jensen, Chris, and Walt Scacchi. "Governance in Open Source Software Development Projects: A Comparative Multi-Level Analysis." In *Open Source Software: New Horizons*, edited by Pär Ågerfalk, Cornelia Boldyreff, Jesús M. González-Barahona, Gregory R. Madey, and John Noll, 130–42. IFIP Advances in Information and Communication Technology 319. Springer Berlin Heidelberg, 2010.
http://link.springer.com/chapter/10.1007/978-3-642-13244-5_11.
- Jonkman, Matt. Interview. Telephone, March 7, 2016.
- Jorgensen, Danny L. *Participant Observation: A Methodology for Human Studies*. SAGE, 1989.
- JP, Vossen. Interview. Telephone, April 2, 2016.
- Kline, Stephen. "Innovation Is Not a Linear Process." *Research Management* July/August (1985): 36–45.
- Koch, S., and G. Schneider. "Implementation of an Annotation Service on the WWW-Virtual Notes." In 8th Euromicro Workshop on Parallel and Distributed Processing, 2000. Proceedings, 92–98, 2000.
- Kopan, Tal. "Cybercrime Costs \$575B Yearly." POLITICO. Accessed April 18, 2016.
<http://www.politico.com/story/2014/06/cybercrime-yearly-costs-107601.html>.
- Kriaa, Mohamed, and Zouhour Karray. "Innovation and R&D Investment of Tunisian Firms: A Two-Regime Model with Selectivity Correction." *The Journal of Business Inquiry* 9, no. 1 (2010): 1–21.
- Kroah-Hartman, Greg, Jonathan Corbet, and 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." The Linux Foundation, 2009.
<http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf>.
- Krogh, Georg von, Sebastian Spaeth, and Karim R Lakhani. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy, Open Source Software Development*, 32, no. 7 (July 2003): 1217–41.

- Lakemond, Nicolette, Lars Bengtsson, Keld Laursen, and Fredrik Tell. "Match and Manage: The Use of Knowledge Matching and Project Management to Integrate Knowledge in Collaborative Inbound Open Innovation." *Industrial and Corporate Change* 25, no. 2 (April 1, 2016): 333–52.
- Lamoreaux, Naomi R., and Kenneth L. Sokoloff. "Long-Term Change in the Organization of Inventive Activity." *Proceedings of the National Academy of Sciences of the United States of America* 93, no. 23 (November 12, 1996): 12686–92.
- Larman, Craig, and Victor R. Basili. "Iterative and Incremental Development: A Brief History." *Computer* 36, no. 6 (June 2003): 47–56.
- Lorenzi, Dario, and Cristina Rossi. "Assessing Innovation in the Software Sector: Proprietary vs. FOSS Production Mode. Preliminary Evidence from the Italian Case." In *Open Source Development, Communities and Quality*, 275:325–31. IFIP International Federation for Information Processing. Boston: Springer, 2008. <http://ifipwg213.org/system/files/Assessing%20Innovation%20in%20the%20Software%20Sector.pdf>.
- Mansfield, Edwin. "Patents and Innovation: An Empirical Study." *Management Science* 32, no. 2 (February 1, 1986): 173–81.
- McConnell, Steve. *Rapid Development: Taming Wild Software Schedules*. 1st ed. Redmond, WA, USA: Microsoft Press, 1996.
- Mcdaniel, Patrick, Jacobus Van Der Merwe, Subhabrata Sen, Bill Aiello, Oliver Spatscheck, and Charles Kalmanek. "Enterprise Security: A Community of Interest Based Approach." In Proc. NDSS, 2006.
- McMillan, Robert. "Red Hat Becomes Open Source's First \$1 Billion Baby." *Wired Enterprise*, March 28, 2012. <http://www.wired.com/wiredenterprise/2012/03/red-hat/>.
- Mickel, Matt. Interview. In Person, December 15, 2015.
- "Microsoft Corporation Global Revenue 2002-2015 | Statistic." Statista. Accessed April 15, 2016. <http://www.statista.com/statistics/267805/microsofts-global-revenue-since-2002/>.
- Mossoff, Adam. "How Copyright Drives Innovation in Scholarly Publishing." SSRN Scholarly Paper. Rochester, NY: Social Science Research Network, April 2, 2013. <http://papers.ssrn.com/abstract=2243264>.

- Murray, Fiona, and Scott Stern. "Do Formal Intellectual Property Rights Hinder the Free Flow of Scientific Knowledge?: An Empirical Test of the Anti-Commons Hypothesis." *Journal of Economic Behavior & Organization* 63, no. 4 (2007): 648–87.
- Neuendorf, Kimberly A. *The Content Analysis Guidebook*. 1st edition. Thousand Oaks, Calif: SAGE Publications, Inc, 2001.
- OECD. "Knowledge-Based Capital, Innovation and Resource Allocation." In *Supporting Investment in Knowledge Capital, Growth and Innovation*, 55–125. OECD Publishing, 2013.
- Oreg, Shaul, and Oded Nov. "Exploring Motivations for Contributing to Open Source Initiatives: The Roles of Contribution Context and Personal Values." *Computers in Human Behavior* 24, no. 5 (September 2008): 2055–73.
- "Oslo Manual: Guidelines for Collecting and Interpreting Innovation Data." OECD, 2014. <http://www.oecd.org/site/innovationstrategy/defininginnovation.htm>.
- Ostrom, Elinor. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.
- . *Governing the Commons: The Evolution of Institutions for Collective Action [...]* [...]. Cambridge [u.a.: Cambridge Univ. Press, 1995.
- Parker, Bill. Interview. Telephone, December 15, 2015.
- Perlroth, Nicole. "Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant." *The New York Times*, September 25, 2014.
- Piva, Evila, Francesco Rentocchini, and Cristina Rossi-Lamastra. "Is Open Source Software about Innovation? Collaborations with the Open Source Community and Innovation Performance of Software Entrepreneurial Ventures." *Journal of Small Business Management* 50, no. 2 (April 2012): 340–64.
- Posner, Richard A. "Transaction Costs and Antitrust Concerns in the Licensing of Intellectual Property." *John Marshall Review of Intellectual Property Law* 325, no. 4 (2004). http://chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=2876&context=journal_articles.
- Rajala, Risto, Mika Westerlund, and Kristian Möller. "Strategic Flexibility in Open Innovation – Designing Business Models for Open Source Software." *European Journal of Marketing* 46, no. 10 (September 14, 2012): 1368–88.

- Raymond, Eric. "The Cathedral and the Bazaar (originally Published in Volume 3, Number 3, March 1998)." *First Monday* 0, no. 0 (October 3, 2005).
- Roberts, Paul F. "The State of Open Source Security." *InfoWorld*, March 26, 2015. <http://www.infoworld.com/article/2901893/security/the-state-of-open-source-security.html>.
- Romer, Paul M. "Endogenous Technological Change." *Journal of Political Economy* 98, no. 5 (October 1, 1990): S71–102.
- Sampson, Rachelle C. "R&D Alliances and Firm Performance: The Impact of Technological Diversity and Alliance Organization on Innovation." *Academy of Management Journal* 50, no. 2 (April 1, 2007): 364–86.
- Schumpeter, Joseph A. "Plausible Capitalism and The Process of Creative Destruction; Chapters 6 & 7." In *Capitalism, Socialism, and Democracy*, 72–86. Harper, 1942.
- Schweik, Charles. "Free/Open-Source Software as a Framework for Establishing Commons in Science." In *Understanding Knowledge as a Commons: From Theory to Practice*. Cambridge, Mass.: MIT Press, 2011.
- . "Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects." *Technology Innovation Management Review*, no. January 2013: Open Source Sustainability (2013): 13–19.
- Schweik, Charles M., and Robert English. "Tragedy of the FOSS Commons? Investigating the Institutional Designs of Free/libre and Open Source Software Projects." *First Monday* 12, no. 2 (February 5, 2007). <http://journals.uic.edu/ojs/index.php/fm/article/view/1619>.
- Shapiro, Robert. "The U.S. Software Industry: An Engine for Economic Growth and Employment." Software and Information Industry Association, 2014.
- Silva, Breno. Interview. Telephone, March 10, 2016.
- Sinofsky, Steven. "The Windows 7 Team." Engineering Windows 7. Accessed April 15, 2016. <https://blogs.msdn.microsoft.com/e7/2008/08/17/the-windows-7-team/>.
- "Snort / Mailing Lists." *Sourceforge*, April 3, 2016. <https://sourceforge.net/p/snort/mailman/snort-devel/?page=418>.

- “Software Developers.” Bureau of Labor Statistics, 2016.
<http://www.bls.gov/ooh/Computer-and-Information-Technology/Software-developers.htm#tab-6>.
- Somaya, Deepak, and David J. Teece. “Transaction Costs and Antitrust Concerns in the Licensing of Intellectual Property.” Working Paper. University of Maryland, University of California at Berkley, 2001.
<http://emlab.berkeley.edu/~bhhall/ipconf/SomayaTeece.pdf>.
- “Sourcefire Announces Record Revenue for Fourth Quarter & Full Year 2012.” Market Wired, February 21, 2013. <http://finance.yahoo.com/news/sourcefire-announces-record-revenue-fourth-210500719.html>.
- Sundström, Per, and Annika Zika-Viktorsson. “Organizing for Innovation in a Product Development Project: Combining Innovative and Result Oriented Ways of Working – A Case Study.” *International Journal of Project Management* 27, no. 8 (November 2009): 745–53.
- Teece, David J. “Profiting from Technological Innovation: Implications for Integration, Collaboration, Licensing and Public Policy.” *Research Policy* 15, no. 6 (December 1986): 285–305.
- “The \$1 Trillion Economic Impact of Software.” Business Software Alliance, June 2016.
http://www.bsa.org/~media/Files/StudiesDownload/Economic_Impact_of_Software_Report.pdf.
- “The Future of Cybercrime & Security: Financial and Corporate Threats & Mitigation.” Hampshire, UK: Juniper Research, May 2015.
<http://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion>.
- “Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense.” MITRE Corporation, 2003. http://terrybollinger.com/dodfoss/dodfoss_html/.
- Vaughan-Nichols, Steven J. “Heartbleed: Open Source’s Worst Hour.” ZDNet. Accessed April 17, 2016. <http://www.zdnet.com/article/heartbleed-open-sources-worst-hour/>.
- von Hippel, E. *Democratizing Innovation: The Evolving Phenomenon of User Innovation*. Cambridge, Massachusetts: The MIT Press, 2005.
- von Hippel, Eric. “Horizontal Innovation Networks—by and for Users.” *Industrial and Corporate Change* 16, no. 2 (April 1, 2007): 293–315.

- von Krogh, Georg, Sebastian Spaeth, and Karim R Lakhani. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy, Open Source Software Development*, 32, no. 7 (July 2003): 1217–41.
- Wallsten, Scott J. "The Effects of Government-Industry R&D Programs on Private R&D: The Case of the Small Business Innovation Research Program." *The RAND Journal of Economics* 31, no. 1 (2000): 82.
- Williams, Sam, and Richard M. Stallman. "Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution," 2010. <http://static.fsf.org/nosvn/faif-2.0.pdf>.
- Witten, B., C. Landwehr, and M. Caloyannides. "Does Open Source Improve System Security?" *IEEE Software* 18, no. 5 (September 2001): 57–61.
- Yin, Robert K. *Case Study Research: Design and Methods*. 5 edition. Los Angeles: SAGE Publications, Inc, 2013.

BIOGRAPHY

Justin M. Novak graduated from Hanover High School, Hanover, Pennsylvania, in 2001. He received his Bachelor of Science and Bachelor of Arts from the University of Pittsburgh in 2006, and a Masters of Public and International Affairs from the University Of Pittsburgh Graduate School Of International Affairs in 2008. He worked for the National Security Agency from 2008 – 2014 and the Pennsylvania State Legislature from 2015 – 2016.