Resilience in Free/Libre/Open Source Software: Do founder decisions impact development activity after crisis events?

Wm Salt Hale

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Arts

University of Washington

2021

Committee:
Benjamin Mako Hill
Kirsten Foot

Program Authorized to Offer Degree:
Department of Communication

University of Washington

Abstract

Resilience in Free/Libre/Open Source Software:
Do founder decisions impact development activity
after crisis events?

Wm Salt Hale

Chair of the Supervisory Committee:
Benjamin Mako Hill
Department of Communication

Free/Libre/Open Source Software (FLOSS) is perhaps the most studied of the information-based communal public goods. In this piece, I conceptualize development resilience and founder decisions as two important characteristics of FLOSS. By asking whether the impact of the latter can be measured through the former, I outline three hypotheses concerning software licensing, crisis events, and the interaction between the two. I then conduct a time series analysis of a novel dataset containing longitudinal measures of software packages from the Debian Project. I determine that while the development activity of all projects declines over time, this decline is slower for protectively licensed projects than permissively licensed ones. I also find that the development activity of FLOSS projects increases after encountering a Debian Security Advisory. Combining these predictors, I find that permissively licensed projects demonstrate more development resilience than protectively licensed ones.

TABLE OF CONTENTS

Page

Introduction 1
Background 3
Public Goods, Peer Production, and FLOSS Scholarship 3
FLOSS License Families 4
Founder Decisions in FLOSS 6
Development Activity and FLOSS Licensing 7
Crisis Events and FLOSS Resilience 8
Method 10
Setting 10
Data 12
Measures 16
Analysis 18
Findings 20
H1 results 20
H2 results 22
H3 results 23
Discussion 26
Empirical Contribution 26
Theoretical Contribution 27
Implications for FLOSS Practitioners 28
Potential Limitations 28
Future Research 29
Conclusion 29

Appendix 30

Bibliography 34

Introduction

There is a thriving ecosystem of free/libre/open source software (FLOSS) projects available today (Lenarduzzi et al., 2020; Li et al., 2021; Sánchez et al., 2020). Rather than keep their source code secret or proprietary, a large number of developers have decided to use software licenses which include terms that legally place the output of their *development activity*, or code, into the commons (Lessig, 2006; von Krogh, 2003). Some developers use *protective* FLOSS licenses which require all derivative work to remain in the public domain, whereas others make use of *permissive* variants that only apply to the original source (Bonaccorsi & Rossi Lamastra, 2003; Gamalielsson & Lundell, 2017; Kapitsaki et al., 2015; Lerner & Tirole, 2005).

Most FLOSS projects are started by a very small number of individuals, often just one (Fitzgerald, 2006; Krishnamurthy, 2005). Of the various choices made early on, some are constitutional decisions, "[the] authoritative rules by which individuals will make future decisions" (Ostrom, 1968). When one of these decisions relates to an ideology, and is made with minimal input, the project may carry an individual's ideological bias. I call these choices *founder decisions* and use an affordance framework (Treem & Leonardi, 2013) to conceptualize them.

In terms of FLOSS licensing, if a project licensed under one of the protective terms were to transition to a more permissive model, all past developers would have to be contacted, and their permission granted before the change could be made (Jensen & Scacchi, 2011). Additionally, there has historically been an expressive divide between the body which approves new *open source licenses* and the progenitor of the *free software movement*, largely based on whether code, once in the commons, should carry this protective term (Stallman, 2009). Thus, I argue that when starting a FLOSS project, the selection of a software license is a founder decision.

These founder decisions send a signal which is received by developers, who then make up the community which builds the FLOSS. Prior research has explored how the license a FLOSS project uses impacts its development activity. Colazo and Fang (2009) and Stewart et al. (2006) found evidence that protective licenses were more effective. While Fershtman and Gandal (2007) and Subramaniam et al. (2009) found that it was the permissively licensed projects which were more active.

Since FLOSS would not exist without the work of programmers, and as *crisis events*, "are unusual occurrences that cannot be predicted but are expected" (Coombs & Holladay, 2010), identifying the resilience, "[the] willingness to remain in spite of variability and change" (Butler et al., 2014), of these communities is important. However, only Raja and Tretter (2012) appears to have studied resilience in the context of FLOSS, and their focus was not directed solely on development.

My work attempts to bridge these conversations; measuring the impact of different types of FLOSS licensing on development activity, furthering the understanding of how crisis events affect software development, and using founder decisions to predict development resilience. Additionally, I contribute to the existing literature by conducting an empirical study, utilizing a larger and more up-to-date pool of projects, to help answer the questions of a FLOSS characteristic which has previously shown mixed-findings. I also expand upon the theory of constitutional decisions, describing a new term, *founder decisions*, and further our understanding of the under-explored theory of *development resilience*. Lastly, I make the practical contribution of a novel dataset, along with the code used to generate it.

To accomplish these goals, I performed a large scale statistical analysis of longitudinal digital trace data from 23,208 software projects within the Debian GNU/Linux distribution. As I hypothesize, I find that while protectively licensed packages initially lead to *lower development activity*, they demonstrate a *higher rate of increase in development activity*, when compared to projects using permissive licenses. Additionally, I find that when projects encounter a crisis event, their *rate of development activity increases*. Finally, I find that packages which have protective licenses demonstrate *lower development resilience* compared to projects that are permissively licensed.

I begin with an overview of the research which has occurred within this space, articulating three hypotheses about the relationship between *FLOSS licensing*, *crisis events*, and *development activity*. I then describe the empirical setting, data and measures, and analytic strategy that I utilize. Finally, I present the results of my statistical modeling, followed with a discussion which includes implications, limitations, and future research directions.

Background

As computers have become more central to civilization, the software that makes them useful has evolved from individual research projects to a thriving ecosystem of developers (Deshpande & Riehle, 2008; Ghosh, 2005; Stallman, 2002). Software does not appear out of thin air, rather, programmers encode logical routines in code that computers run, returning with the (hopefully) desired outcome. While some developers and companies keep their source code secret or proprietary, a large number utilize FLOSS licenses, enabling others to assist in the development activity (Bessen, 2005; Bonaccorsi & Rossi Lamastra, 2003; Sen et al., 2011; von Krogh, 2003).

These licenses outline, "the legal agreement of how the project is developed and distributed" (Viseur & Robles, 2015), and all contributed code is generally subject to their terms. What makes these licenses different is that once software has been released under a FLOSS license, "[it] sits in the commons. Anyone can take it and use it as she wishes. Anyone can take it and come to understand how it works. [...] Everything is public; anyone, without having to seek the permission of anyone else, may join the project" (Lessig, 2006).

In other words, developers who decide to contribute to FLOSS licensed projects are explicitly giving up the claim to their intellectual property which has been codified as knowledge good. Instead, their work is being released to the public domain, enabling the creation of future peer-produced knowledge commons.

Public Goods, Peer Production, and FLOSS Scholarship

In economics, the term "public good" describes something which is both non-excludable (in the public domain or commons) and non-rivalrous (the use by one does not diminish another's usage) (Romer, 1990). Fulk et al. explored how such goods operate within interactive communication systems, outlining explicit information and communicative public goods. These differ from material public goods in that their very nature precludes excludability and rivalrousness. Additionally, two classes of such goods were teased out, based on the features of connectivity (direct communication links to other members) and communality (being jointly held and sustained by all members) (Fulk et al., 1996).

Benkler (2006) achieved further distinction by following a line of inquiry concerning the creation of these information-based communal public goods. Commons-based peer production (CBPP) is, "a form of open creation and sharing performed by groups online that: set and execute goals in a decentralized manner; harness a diverse range of participant motivations, particularly non-monetary motivations; and separate governance and management relations from exclusive forms of property and relational contracts (i.e., projects are governed as open commons or common property regimes and organizational governance utilizes combinations of participatory, meritocratic and charismatic, rather than proprietary or contractual, models)" (Benkler et al., 2015). And, "the best-known examples of commons-based peer production are the tens of thousands of successful free software projects that have come to occupy the software development market" (Benkler & Nissenbaum, 2006).

Many interdisciplinary scholars have studied FLOSS development. They have focused on motivation (Hars & Ou, 2002; Hertel et al., 2003; Lakhani & Wolf, 2005; Lerner & Tirole, 2002; Stewart & Gosain, 2006), organization (de Laat, 2007; Kogut & Metiu, 2001; O'Mahony & Ferraro, 2007; von Hippel & von Krogh, 2003), production (Crowston et al., 2003; Deshpande & Riehle, 2008; von Krogh, 2003), along with numerous other interconnected topics (Androutsellis-Theotokis et al., 2011; Crowston et al., 2008).

One of the most heavily studied phenomena within this field is the impact of the licenses themselves. German and Hassan (2009) addressed "license mismatches" due to incompatible legal requirements. Whereas German et al. (2010) developed a method to automatically identify FLOSS licenses in source code. Jensen and Scacchi (2011) delved into the process of a project changing its license. Viseur and Robles (2015) examined the motivations and impacts of changing a project's license. And Kapitsaki et al. (2015) performed a comparative analysis of license usage and characteristics across FLOSS. Recently, Gamalielsson and Lundell (2017) explored the relationship between which FLOSS license was used and whether additional conditions were placed upon contributions.

FLOSS License Families

The academy's interest in licensing is largely driven by differences in the legal term which specifies whether code that builds upon, or derives from, the original piece of FLOSS must carry the same software licensing

terms (Bonaccorsi & Rossi Lamastra, 2003; Gamalielsson & Lundell, 2017; Kapitsaki et al., 2015; Lerner & Tirole, 2005; Sen et al., 2008). One side argues that these terms are protective, ensuring that all related work remains in, thus expanding, the commons. On the other hand, some argue that if the point is to make the code public, it should be free of any and all restrictions, including what licenses must be applied to future derivatives.

The Institute for Legal Questions on Free and Open Source Software (ifrOSS) has categorized 208 of these licenses into one of five *license families*.[1] The main selection criteria used is the distinction regarding whether or not the license carries a protective term, the legal guarantee that code based on the original FLOSS must also be protectively licensed. Within this divide, ifrOSS provides nuance between strict and limited versions of the term and its counterpart. They also define a fifth category of licenses, "with particular privileges", which are often used in the event that a company is transitioning their proprietary software to FLOSS.

Given that the original ifrOSS category names are translated from German, I will be using my own naming schema for clarity. Thus, the five FLOSS license families are: protective, limited-protective, limited-permissive, permissive, or post-commercial. The description of these license families, along with examples of specific licenses, terms, and projects, can be found in Table 1 on page 15.

This list is valuable, not only because it elucidates an otherwise subtle difference between FLOSS licenses, but also because it is generated by an organization other than the Free Software Foundation (FSF) or Open Source Initiative (OSI). These two organizations are the primary sources that outline what software license terms qualify as supporting "free software" or "open source", and each maintains an up-to-date list of FLOSS licenses that they approve of.[2][3][4]

Historically, there has been a significant divide between these two bodies; the FSF being the *free soft-*

---

[1] This count is as of 2020-10-01 and does not include 90 licenses which have been found in the FLOSS ecosystem, but use terms which do not align with either the Free Software Definition or Open Source Definition. An archived version of the ifrOSS page can be viewed using the Internet Archive Wayback Machine: https://web.archive.org/web/20201001144451/https://ifross.org/en/license-center

[2] See the definitions linked in footnote 1.

[3] FSF commentary on various software licenses: https://www.gnu.org/licenses/license-list.html

[4] List of software licenses that have been approved by the OSI: https://opensource.org/licenses/

ware movement progenitor, disapproving of the pragmatic nature of the OSI (Stallman, 2009). Or, as Meretz (2007) puts it, "OSI [and] GNU – brothers and sisters at daggers drawn." I consider this conflict largely being based on whether code, once freed, should carry the protective term.

Founder Decisions in FLOSS

Another large body of scholarly work looks at founders, the people who start communities, organizations, and projects on the Internet (Foote & Contractor, 2018; Kraut & Fiore, 2014; Kraut et al., 2012; Schweik & English, 2012). Of the many choices which need to be made before developing software, some can be considered constitutional decisions, "[the] authoritative rules by which individuals will make future decisions" (Ostrom, 1968). Ostrom explains that these choices are differentiated by two specific properties. First, they have the potential to impact every part of the project or organization. Second, once made, they are extremely difficult to change (1968).

Most FLOSS projects are started by a very small number of individuals, often just one (Krishnamurthy, 2005). This implies that minimal voices are involved when constitutional decisions are being made (Fitzgerald, 2006; Fogel, 2005; Gasparini et al., 2020). Thus, these founders wield great power over the project's future, and that power is clearly enacted when a choice ends up being a constitutional decision.

When a constitutional decision is codified into a shareable artifact, the properties of broadness of impact and resistance to change are crystallized, and may be thought of as affordances. Despite the "significant inconsistencies in use of the term" (Evans et al., 2016), affordances were first conceptualized as, "what things furnish, for good or ill" (Gibson, 1966, p. 285). Of the numerous ways this has been taken, Treem and Leonardi point out that, "the most nuanced writings on the relationship between technology and organizational change emphasize the relational character of affordances." And that, "materiality exists independent of people, but affordances do not" (2013).

Finally, certain choices are more closely tied to one's ideology. Because of the aforementioned affordances of constitutional decisions, when a selection of this nature is made, all subsequent work may carry an ideological bias. I call this category of choices *founder decisions* and suggest that they have a third affordance, the ability to carry a founder's imprint.

Returning to the discussion of FLOSS licensing, if a project licensed under one of the protective terms were to transition to a more permissive model, all past developers would have to be contacted, and their permission granted before the change could be made (Jensen & Scacchi, 2011). Additionally, it is the very nature of software licensing to impact the entire codebase and, in the case of protective licenses, beyond (Viseur & Robles, 2015). Thus, when starting a FLOSS project, the selection of a software license should be considered a founder decision.

While a FLOSS project may initially only have one developer, key benefits of using these licenses, such as lessening the burden on any individual programmer, are only realized by attracting others (Ghosh, 1998; Raymond, 1999; Weber, 2004). Founder decisions, such as license choice, send a signal which is then received by certain types of developers, likely ones that have some alignment with the ideological imprint being transferred. These developers make up the project's community, providing their time and energy to the creation of the FLOSS.

Development Activity and FLOSS Licensing

Aside from the research into FLOSS licenses discussed prior, a major question has been what effect licensing has on development activity. Stewart et al. conducted an analysis of 138 projects sampled from Freshmeat, an information hub for FLOSS projects. However, they were unable to find statistical support that, "restrictive [i.e. protective] licensing elicits greater development activity" (Stewart et al., 2006).

Fershtman and Gandal collected a sample of the 71 most active projects of January 2000 on SourceForge, at the time, the largest web hosting platform for FLOSS. They found, "that output per contributor in open source projects is much higher when licenses are less restrictive and more commercially oriented" (Fershtman & Gandal, 2007).

Colazo and Fang returned to the question, also using SourceForge to collect a sample of 62 projects with at least 2 years of coding activity. They found statistically significant support for "a higher level of coding activity" and "project speed" in copyleft (i.e. protectively) licensed FLOSS projects (Colazo & Fang, 2009).

Subramaniam et al. again returned to SourceForge, compiling a dataset of 8,627 projects that had both complete information, and been registered between January 1999 and December 2005. They found, "that

restrictive licenses (Strong-Copyleft and Weak-Copyleft licenses) negatively impact the activity levels [...] only if the target audiences are software developers" (Subramaniam et al., 2009).

My work contributes to this conversation in a number of ways. First, by compiling, and making use of, a large dataset containing projects from a platform which does not exist solely to host FLOSS, I hope to target software that is actively in use, avoiding projects which are nascent or merely hobbies. Furthermore, rather than using sampling methods or fixed time-range panel data, I am able to look at the entire lifespan of FLOSS projects as found on the platform. Additionally, I utilize a more impartial body to categorize licenses, rather than relying upon the oftentimes misaligned originators of these classifications. Finally, I apply linear mixed-effect regression models, controlling for unseen heterogeneity, while focusing on a specific measure of development activity, as outlined by Colazo and Fang (2009).

As the protective term exists to ensure development that goes into FLOSS will remain a public good in subsequent projects (Lessig, 2006), protectively licensed projects may be more attractive to programmers associated with the free software movement. These developers are likely more ideologically driven than those who merely want to ensure that their direct development work is placed in the commons (Stewart & Gosain, 2006). This commitment to a cause may be demonstrated through action. Thus, FLOSS developers may spend more time developing projects that have a protective license family license, leading to increased development activity. Therefore, I hypothesize: (**H1**) FLOSS projects with *protective* licenses will have more *development activity*, compared to those with *permissive* licenses.

Crisis Events and FLOSS Resilience

Returning to FLOSS research more broadly, an area which has somehow avoided almost all attention is *resilience*, "the ability to recover from a disturbance" (Raja & Tretter, 2012). Counter to the free software movement's utopian ideals which envision a "post-scarcity world" (Stallman, 1984), where "people could take the freedom to cooperate as they please" (Merten, 2004), and everything just works since "given enough eyeballs, all bugs are shallow" (Raymond, 1999), occasionally problems do occur. Sometimes these are small scale bugs, perhaps causing an annoyance. Other times these issues can lead to a full-blown crisis event.

The Handbook of Crisis Communication defines a crisis as, "the perception of an unpredictable

event that threatens important expectancies of stakeholders and can seriously impact an organization's performance and generate negative outcomes" (Coombs & Holladay, 2010). While Coombs unpacks this further than I will here, there are two key points I want to highlight. First, based on the understanding that, "meaning is socially constructed," external *perception* is required. Second, an important distinction between a crisis and annoyance comes down to whether there is a "threat" of "negative" "impact" greater than the "expectancies".

Examples of crises that are occasionally seen in the FLOSS ecosystem include maintainer transitions, forking events, and security announcements. In terms of transitions, when a core project contributor passes on the reins, the community may question whether the newcomer will be as effective. Forking on the other hand occurs when a vocal portion of the community decides to split off, starting a competing version of the FLOSS project, based on the original code. Finally, noteworthy security vulnerabilities may forewarn an unsteady foundation or other unseen risk factor.

All of these examples provide an opening for the community to question whether they are focusing their energy in the correct place. While core contributors may continue chugging along, peripheral developers are likely less committed. For instances, forking events force programmers to decide which project to contribute their future work toward. A newcomer may see both and have to decide between the one with a longer history or the other with more recent activity.

Thus, I theorize that after a crisis event occurs, some projects may fall apart, while other communities may rally. I call projects and communities that bounce back resilient due to their, "willingness to remain in spite of variability and change" (Butler et al., 2014). In the software development space, crisis events are almost exclusively studied in terms of what went wrong (Durumeric et al., 2014; Walden, 2020; Yilek et al., 2009). Only Raja and Tretter (2012) appear to have studied resilience in FLOSS. Even that was only one component of a three part "viability index," and not directly focused on *development resilience*. This is an important area to explore as crisis events, "are unusual occurrences that cannot be predicted but are expected" (Coombs & Holladay, 2010).

Because FLOSS projects are built by individuals with varied motivations (Lakhani & Wolf, 2005; Lerner & Tirole, 2002), the more developers who take notice, the more likely development will occur (Yunwen Ye & Kishida, 2003; Zimmermann & Casanueva Artís, 2019). Given that crisis events, by defi-

nition, lead to increased attention, this may lead to an increased pool of potential developers, leading to an increase in development activity. Walden (2020) pointed out that this process occurred after the 2014 OpenSSL vulnerability, Heartbleed. As a result, I hypothesize: (**H2**) A *crisis event* will increase the rate of *development activity* in FLOSS projects.

Presuming H1 and H2 are shown: protectively licensed projects' development activity being measurably different from those in the permissive family; and FLOSS projects' development activity demonstrating resilience after crisis events. Then, because of the imprint carried by the founder decision, a difference in the level of development resilience between FLOSS projects of differing license families may be visible. Following this logic, and given the ideological foundation of protective licenses, I hypothesize: (**H3**) FLOSS projects with *protective* licenses will demonstrate increased *development resilience* after a *crisis event*, compared to those with *permissive* licenses.

The conceptual model in Figure 1 on the following page demonstrates how *founder decisions* impact both *development activity* and the relationship between *development activity* and a *crisis event*. It also demonstrates the effect of *project age* on *development activity*. Next, I outline the empirical setting, data, and measures.

Method

Setting

I test my hypotheses using a dataset drawn from the Debian Project.[5] Debian, launched in August 1993, is a GNU/Linux distribution which only accepts software released under FLOSS licenses (Project, 2004). They track nearly 70,000 projects, some with changelogs spanning over 25 years (Zacchiroli, 2015). What's more, they have been announcing software security vulnerabilities over the same period of time. Due to the strong FLOSS ethos that Debian ascribes to (Coleman, 2005; Coleman & Hill, 2004), all of this information is made publicly available (Nussbaum & Zacchiroli, 2010).

Given the historic impact, quantity, and lack of fees associated with this data, many scholars have studied Debian. As might be expected, plenty of research has focused on the developers (Krafft et al.,

[5] https://debian.org/

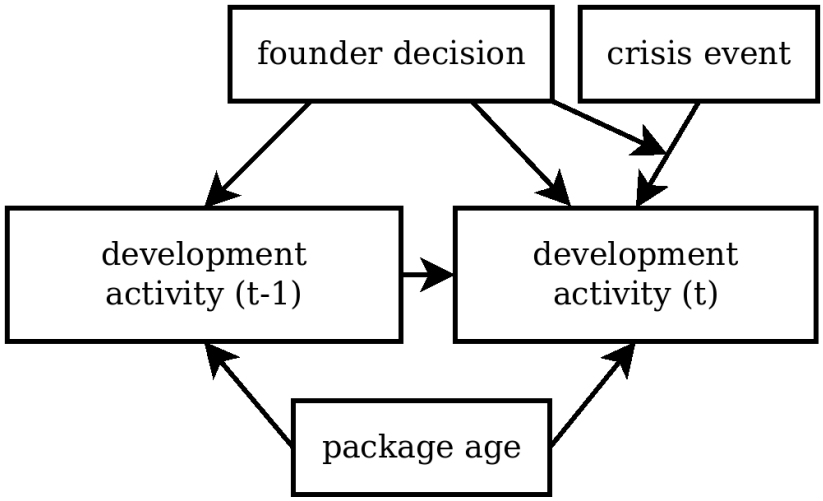

Figure 1: The conceptual model grounding H1 to H3 on pages 8–10. It demonstrates the interactions between *development activity*, *project age*, *founder decisions*, and *crisis events*. Package age and founder decisions both have a direct effect on all development activity. Crisis events impact development activity after they occur. Founder decisions modify the relationship between crisis events and development activity.

2016; Michlmayr & Senyard, 2006) and their code (Champion & Hill, 2021; Chen & Wagner, 2007; Claes et al., 2015; Galindo Duarte et al., 2010; Nguyen & Holt, 2012), as well as the security vulnerabilities (Yilek et al., 2009), community (Mateos-Garcia & Steinmueller, 2008; Robles et al., 2005; Sadowski et al., 2008), and even meta questions regarding conducting research on Debian (Herraiz et al., 2011; Spaeth et al., 2007).

The Debian Project release cycle is such that there are always three actively maintained distributions, "stable", "testing", and "unstable".[6] Each of these releases makes use of a codename, as one distribution will move through all three stages.[7] A specific distribution links to a similarly named repository of software packages. The version of a package in one Debian distribution may vary from another, and while there is advice on selecting between them, the general recommendation is to, "stick with the stable distri-

[6] https://www.debian.org/releases/

[7] https://www.debian.org/doc/manuals/debian-faq/ftparchives.en.html#codenames

bution".[8]

In terms of the software projects themselves, the Debian repositories contain both compiled binaries and raw source code. Source packages[9] contain all of the necessary files to compile a project. Whereas binaries may contain code from a number of different projects that have already been pre-compiled together. As I am interested in the development process of individual projects, I focus on the *source packages*. To compare within distribution differences, I select the package version available in the *stable* repository.

Data

To construct my dataset, I wrote a script with the Scrapy web crawler[10] which downloaded the full list of source packages available in the Debian 10 ("Buster") repository on October 1, 2020. Every package hosted on Debian has a public webpage which provides a description and links to a number of relevant resources, including metadata files containing the project changelog and copyright information.[11]

The changelog file provides a detailed update of every version of the software which was added to the Debian repository. Each entry provides the package name, version number, target Debian release, urgency, a brief description of changes since the previous version, the name and email of the maintainer who uploaded this version, and a timestamp of when the change was added to Debian.

The copyright file provides details about the software license terms which apply to various parts of the package. While there has been a proposed specification[12] for machine-readability of this file, application of this policy is optional and the copyright files are extremely inconsistent. Much of the metadata the copyright-format specification expects is also optional, but the required entries include: what source files are covered by which license, and under who's copyright authority.

A separate set of project related metadata that is hosted by the Debian Project includes a series of

---

[8] https://www.debian.org/doc/manuals/debian-faq/choosing.en.html

[9] https://wiki.debian.org/Packaging/SourcePackage

[10] https://scrapy.org/

[11] https://packages.debian.org/

[12] As of 2020-11-17, this has been accepted as a standard and can be found at https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/

public vulnerability notices, or Debian Security Advisories (DSAs).[13] These are condensed subsets of information from the Debian Security Tracker,[14] where common vulnerability and exposure (CVE) information is gathered about all packages, whether or not the Debian version is implicated. Each DSA contains a unique identifier, date reported, affected package name, and additional details regarding the security concern. I was able to download the full list of DSAs, dating back to pre-1997, with my Scrapy script.

Making use of the output from dpkg-parsechangelog,[15] I was able to gather information from the raw changelog files into a formatted dataset. During this process, all timestamps were converted to the POSIX date standard.[16] This ended up being non-trivial given that, while Debian policies specify a standard format which should be followed,[17] changelog dates were occasionally recorded in a free form way, or lacked some sort of precision such as timezone. This was also the case for dates associated with DSAs.

Project copyright files were also quite difficult to work with given a general lack of standardization or stylistic patterns. However, I was able to identify and extract license names using a custom Python script based on an aged dpkg-licenses tool.[18] I began by parsing all files that were machine readable. If this was not the case, I used a series of regular expressions in an attempt to match the applicable license or licenses with a known pattern. During this process, all licenses underwent a text normalization process for future comparison.

Utilizing the ifrOSS categorization schema, a hand-generated lookup table of license families, the extracted copyright file data, and Pandas (The pandas development team, 2021; Wes McKinney, 2010), I was able to identify the license family of 23,211 projects. See Table 1 on page 15 for an example of the licenses and terms which each family includes. Projects that had multiple licenses were checked, and if all licenses shared a family, this was captured. I also used Pandas to identify whether a project had ever

[13] https://www.debian.org/security/#DSAS

[14] https://security-tracker.debian.org/tracker

[15] https://wiki.debian.org/Teams/Dpkg

[16] https://pubs.opengroup.org/onlinepubs/9699919799/utilities/date.html

[17] https://www.debian.org/doc/debian-policy/ch-source.html#debian-changelog-debian-changelog

[18] https://github.com/daald/dpkg-licenses

encountered a DSA, and if it had, how many DSAs were seen prior to an individual changelog entry.

Before transitioning from Python to R (Team, 2021),[19] I combined the changelogs, license families, and DSAs into a single dataset using Pandas. This generated my *unit of analysis*: a single row for each project changelog entry. The combined dataset was saved as a comma-separated values (CSV) file for import by R.

By combining various tools from the tidyverse R library (Wickham et al., 2019), a number of new variables were created and the dataset further refined. `change_delta_days` was generated by taking the difference between an entry's `change_date` and the previous one, within the same project as specified by `package_name`. During this process, 3,429 changes in 1,494 projects which appeared before the previous one's timestamp were removed as they violated our understanding of time as continuous.

Another new variable, `seen_dsa`, identifies whether or not a DSA had been issued for the package before any given changelog entry, as specified by `change_date` and `change_number`. During this process, 17 projects which had been the subject of a DSA prior to one of their changelog entries were removed as the impact of the announcement could not be compared to a state before the DSA was released.

The license families as extracted contained a number of inconsistencies that were addressed. 3,660 projects which belonged to multiple incompatible[20] license families and projects which had been coded as unknown due to not having parsable license files, had their license family marked as *other*. An additional 1,639 projects which had licenses that were parsed but uncategorizable, due to being proprietary or inconsistent with the ifrOSS schema, also had their license family marked as *other*. Finally, a new variable, `license_family_combined`, was created to cluster licenses which belonged to either the primary or limited variant of a license family and 3 projects that had post-commercial licenses were combined with the projects that were marked *other*.

---

[19] While Pandas allows for statistical modeling, R is much more suited for the modeling process. Additionally, conducting vector operations on matrices is something that is frequently done within R and the tools for further processing were much faster.

[20] In the sense that they could not be placed within a single category, not that they were legally incompatible. See German and Hassan (2009) for discussion about FLOSS licensing mismatches.

Table 1: "Open Source Licenses" as categorized by the Institut für Rechtsfragen der Freien und Open Source Software (ifrOSS). I have utilized my own more consistent naming schema while including the original English translation in parentheses under each description. Full listing available at the ifrOSS License Center, https://ifross.org/en/license-center

License Family (ifrOSS description)	example licenses	example provisions	example packages
Permissive (without copyleft)	BSD, Apache, MIT, Python, Sendmail	Users may change the license; Package may be enhanced and the enhanced version commercialized	GraphViz, ImageMagick, iSpell, X-ScreenSaver, ZShell
Limited-Permissive (restricted choice)	Artistic, LaTeX, Microsoft Public License	Distribution may be restricted; Redistribution rights may vary depending on type of modifications	ack, LaTeX, mailagent, npm, Perl
Limited-Protective (restricted copyleft)	LGPL, CDDL, Mozilla, NASA, BitTorrent	A different license may be used for changes placed a separate file; Often used when combining projects with multiple licenses	Firefox, LibreOffice, Pulse Audio
Protective (strong copyleft)	GPL, European Union, IBM Public	Users may not change the license; All additions must be distributed under a similar license as the original	GNU/Linux, memcached, Ruby, SSL
Post-Commercial (privileges)	Apple Public Source, Netscape, RealNetworks	Alterations may be subject to special privileges of the original author; Often used to turn proprietary software open source	hevea, spamprobe

I began with 28,841 projects and 548,219 unique changelog entries. I dropped 66 projects due to data availability or parsing issues. As a result, my analytic dataset contained 28,775 projects, of which 944 had experienced a DSA. This dataset included a total of 534,473 unique changelog entries. 12,389 had their combined license family categorized as *protective*, 10,819 as *permissive*, and 5,567 as *other*. I have published the full dataset, along with the code used to build this dataset, in a replication archive on the Harvard Dataverse.[21]

Measures

The changelog data that I just described spans a project's incorporation into Debian through October 1, 2020. My unit of analysis, the changelog entry, is an observation of every code update committed to the Debian version of a project's source package. Age, as outlined by Stewart et al. (2006), can serve as a proxy for many underlying factors important to development activity. Thus, every entry within a project has been coded with a sequential numeric identifier, `change_number`, to indicate the project age at the time of the change.

I follow in the footsteps of Colazo and Fang (2009) and measure development activity, "project speed", via inter-release time. This is the number of days between a changelog entry and the entry before it, within the same package. If a project is being developed quickly, there will be small periods of time between changelog entries, whereas an inactive project will have longer periods between each changelog entry. This has been coded as `change_delta_days` in the data, which is the difference in `change_date` between this entry and one entry lagged. The first change of every project has been set to 0.

Given the impact, adhesion, and limited voices which go into founder decisions, I make use of the project's software license as an indicator of one such early choice. This is useful, given that a clear divide exists between two categories, or families, of licenses: protective or permissive. To measure this founder decision, I consider whether or not a project's license family `is_protective`. The boolean variable is coded as 1 if the license family is protective and 0 if it is not.

---

[21] See Appendix on page 30 to obtain the data and code.

Table 2: Mapping of concepts to operationalized variables of interest as found in the Debian Project data. `change_delta_days` is the dependent variable for all three linear mixed-effect models 1 to 3 on pages 19–20.

concept	conceptual definition	variable	variable definition	operationalization
development activity	The amount of technical energy a pool of developers are putting into a project at any given time	change_delta_days	difference between changelog entry time and previous entry commit time	Continuous (measured in days)
package age	The amount of time which has passed since a project was first released publicly where others could download and contribute to it	change_number	number of code updates in changelog before entry	Count
crisis event	A situation, which becomes public, that threatens an audience's faith in the reliability of a project	seen_dsa	whether or not there was a security announcement prior to changelog entry	Boolean (1 = encountered dsa)
founder decision	A choice made by the founder(s) of a project which has wide impact and is difficult to change once others are involved	is_protective	whether or not a package license is of the protective license family	Boolean (1 = protective)
development resilience	The degree to which a project can recover from a crisis event	n/a	difference between change_delta_days before and after a crisis event	Continuous (measured in days)

Finally, as resilience is the measure of whether a community can recover from a moment of instability, it is important to identify those moments, or crisis events. I use the presence of a DSA as a marker for when an issue, which is of a magnitude that it may impact the sustainability of a project, has been brought to the community's attention. To indicate whether a changelog entry occurred before or after a DSA, each change has been coded with the boolean variable `seen_dsa` which is equal to 0 if it has a timestamp, or `change_date`, before the DSA, and 1 afterward. Code updates made before this point may then be compared to those uploaded after the discontinuity event.

See Table 2 on the previous page for a summary of how the concepts discussed in the background section map onto each of these measures. Next, I describe my analysis and the statistical models that I fit, followed by the results.

Analysis

To test my hypotheses, I conduct a time series analysis of longitudinal changelog data, searching for the impact of *founder decisions* and *crisis events* on a project's *development activity*, as moderated by the project's *age*.

Due to the rapid development pace of some software, compared to the maintenance state that others reach, the time between changes can vary widely, both within a project and across different ones. Similarly, some FLOSS projects are very complex or mature, and include many changes, while others may be new or straightforward, and only have a handful. I account for this skew by applying a log-transformation to both `change_delta_days` and `change_number`, so that their values more closely follow the Gaussian standard normal distribution.

Additionally, as development activity is a repeated measure within individual packages, there is the risk of unobserved heterogeneity. To account for the potential of this threat causing serial correlation of standard errors, I include a random effects parameter based on `package_name`, described in Gelman and Hill (2007, Ch. 12) as the "partial-pooling" of estimates or a "varying-intercept" model. This is accomplished by using the *lmer* function that comes with the lme4 R library (Bates et al., 2015) to compute my linear mixed-effect regression models.

Models

To test **H1**, I make use of an indicator representing the *founder decision* of license selection, `is_protective`. As they do not relate to my variable of interest for this hypothesis, I remove all projects with a license family of *other*.[22] The following linear mixed-effect regression model is used to test the relationship described in H1:

$$\begin{aligned} y_{\text{log(change\_delta\_days)}} =& \beta + \beta \cdot \text{log(change\_number)} + \beta \cdot \text{is\_protective} \\ &+ \beta \cdot (\text{log(change\_number)} \times \text{is\_protective}) \\ &+ \alpha \cdot 1|\text{package\_name} + \varepsilon \end{aligned} \tag{1}$$

For **H2**, I introduce longitudinal security notification data through the variable, `seen_dsa`. Using an interrupted time series (ITS) analysis, I compare the average development activity of a project before and after it encounters a *crisis event*. Thus, the linear mixed-effect regression model for testing H2 is:

$$\begin{aligned} y_{\text{log(change\_delta\_days)}} =& \beta + \beta \cdot \text{log(change\_number)} + \beta \cdot \text{seen\_dsa} \\ &+ \beta \cdot (\text{log(change\_number)} \times \text{seen\_dsa}) \\ &+ \alpha \cdot 1|\text{package\_name} + \varepsilon \end{aligned} \tag{2}$$

Lastly, **H3** is tested with an ITS analysis which includes both measures, *founder decision* and *crisis event*. As was the case for Model (1), I remove all projects with a license family of *other*[23] and test H3 with the following linear mixed-effect regression model:

---

[22] This is done to reduce noise around the interaction. I test this relationship using the complete dataset and the findings do not change. Results can be found in the online supplement.

[23] See footnote 24 on page 22 for more details.

$$
\begin{aligned}
y_{\text{log(change\_delta\_days)}} = & \beta + \beta \cdot \text{log(change\_number)} + \beta \cdot \text{seen\_dsa} + \beta \cdot \text{is\_protective} \\
& + \beta \cdot (\text{log(change\_number)} \times \text{seen\_dsa} \times \text{is\_protective}) \\
& + \alpha \cdot 1|\text{package\_name} + \varepsilon
\end{aligned} \tag{3}
$$

Findings

Table 3 on the next page provides a summary of results from the linear mixed-effect regression models (1), (2), and (3) testing the relationships described in H1, H2, and H2 on pages 8–10. Due to the log-transformed parameters, subset data, and varied measures, interpreting these coefficients directly can be challenging. In order to facilitate interpretation, I present the findings as applied to prototypical projects, generated from model-predicted values, in Figures 2 to 4 on pages 23–25.

I find statistically significant changes in the rate of development activity throughout all three models. Protective licensing and DSAs each increase the rate individually, providing support for H1 and H2. However, protectively licensed projects initially have decreased development activity, when compared to permissive licensing, leading to mixed findings for H1. Additionally, taken together, the development activity of protectively licensed projects appears to be less resilient to crisis events than that of permissively licensed ones, a relationship which is opposite of what I described in H3.

H1 results

Model (1) shows mixed support for H1, that protective licenses lead to increased development activity, when compared to permissive licenses. The average time between changelog entries for protectively licensed projects is 18.51 days, while permissively licensed projects generally release a change every 15.09 days. Clearly, these results suggest, contrary to H1, that permissively licensed projects have more development activity. However, this split is heavily influenced by the difference of intercept between the families. For example, the first change of a project carrying a protective license may be made after 10.23 days, whereas a similar permissively licensed project might have its first code update after 6.67 days (β =0.43, β 95%CI =0.39 – 0.47, p = 0.02).

	Model (1)	Model (2)	Model (3)
package (*intercept*)	1.90***	2.21***	1.83***
	(0.02)	(0.01)	(0.02)
log(change_number)	0.82***	0.68***	0.86***
	(0.01)	(0.00)	(0.01)
is_protective	0.43***		0.45***
	(0.02)		(0.02)
log(change_number) × is_protective	−0.22***		−0.24***
	(0.01)		(0.01)
seen_dsa		1.06***	1.34***
		(0.07)	(0.16)
log(change_number) × seen_dsa		−0.31***	−0.47***
		(0.01)	(0.04)
seen_dsa × is_protective			−0.28
			(0.18)
log(change_number) × seen_dsa × is_protective			0.18***
			(0.04)
AIC	1749815.82	2121272.94	1748947.93
BIC	1749881.84	2121340.08	1749057.96
Log Likelihood	−874901.91	−1060630.47	−874463.96
Number of Changelog Entries	443777	534462	443777
Number of Packages	23208	28775	23208
Package Variance *(Intercept)*	0.79	0.85	0.79
Residual Variance	2.81	2.88	2.80

$^{***}p < 0.001; ^{**}p < 0.01; ^{*}p < 0.05$

Table 3: Linear mixed-model regression summaries of Models 1 to 3 on pages 19–20.

That said, when comparing the *rate* of development activity, I find that the pace protectively licensed projects are releasing changes is only increasing by 1.81 days, while permissively licensed projects are increasing the time between updates by 2.26 days (β =-0.22, β 95%CI =-0.24 – -0.21, p = 0.008). In other words, permissively licensed projects initially have more development activity, yet protectively licensed projects decrease their rate of development activity over time 19.99% slower.

This can be seen in Figure 2 on the following page where two prototypical projects, with 50 changelog entries each, differing only on license family, have been generated. After predicting and plotting a value for each of the code updates with Model (1), a line representing the average development activity is placed atop each project. The results described can be seen in these lines, both in the shift of intercept and slope. As this model demonstrates, holding all other factors constant, the protectively licensed project ends up having increased development activity shortly after the first commit, supporting H1.

H2 results

Model (2) supports H2, that a crisis event increases the rate of development activity (i.e. FLOSS demonstrates resilience). Prior to a DSA, the pace between changes projects made was increasing by 1.98 days on average. After experiencing a DSA, generally the pace between changes was only increasing by 1.45 days (β =-0.31, β 95%CI =-0.34 – -0.29, p = 0.015).[24] In other words, the rate of development activity speeds up by 26.99%. Although potentially interesting, the change in intercept is not interpreted as it does not relate to my hypothesis.

I demonstrate these results in Figure 3 on page 24, by generating a prototypical project which has 50 changelog entries. After 25 changes, the project encounters a DSA on the 26th, while all other attributes are held constant. The development activity of this artificial project is predicted using Model (2) then graphically represented with the ggplot2 R library (Wickham, 2016). The plot is stratified by changelog entries before and after the DSA, and a line displaying the average development activity overlays each series. The difference in slope of this line, is a visual representation of the findings which support H2.

[24] Confidence intervals are being listed due to the distribution's asymmetric nature which may lead to highly skewed standard errors. Instead, I make use of a profiled deviance function as suggest in Bates et al. (2015), based on the examples from: https://github.com/lme4/lme4/issues/497

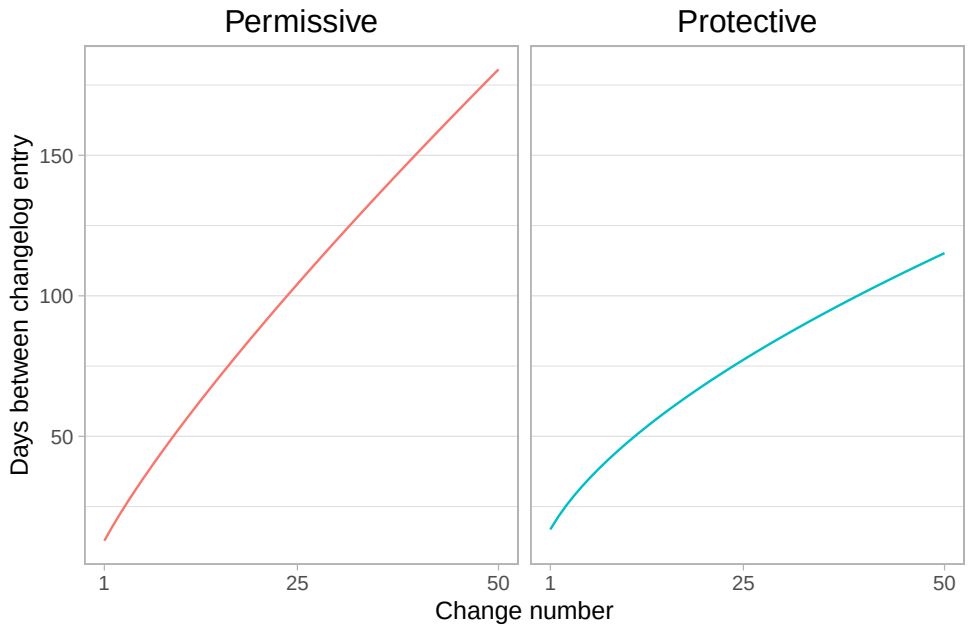

Figure 2: Two prototypical projects with 50 changelog entries each, differing only by their license family (Left: Permissive, Right: Protective). Each integer along the X-axis represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`). Coordinates for the artificial projects were predicted by Model 1, making use of *rust-string*, the project with the median random effects for the `package_name`.

H3 results

Model (3) does not support H3, that development activity of protectively licensed projects is more resilient to a crisis event than permissively licensed ones. Rather, I find that while protectively licensed projects maintain a higher average rate of development activity after a DSA, permissively licensed ones demonstrate a larger difference. For instance, before a DSA, the pace protectively licensed projects are releasing software updates is increasing by 1.85 days on average. After experiencing a DSA, the pace these same projects are releasing new code is increasing by approximately 1.39 days. In comparison, projects carrying permissive licenses have an increasing rate of about 2.35 days between changelog entries before being

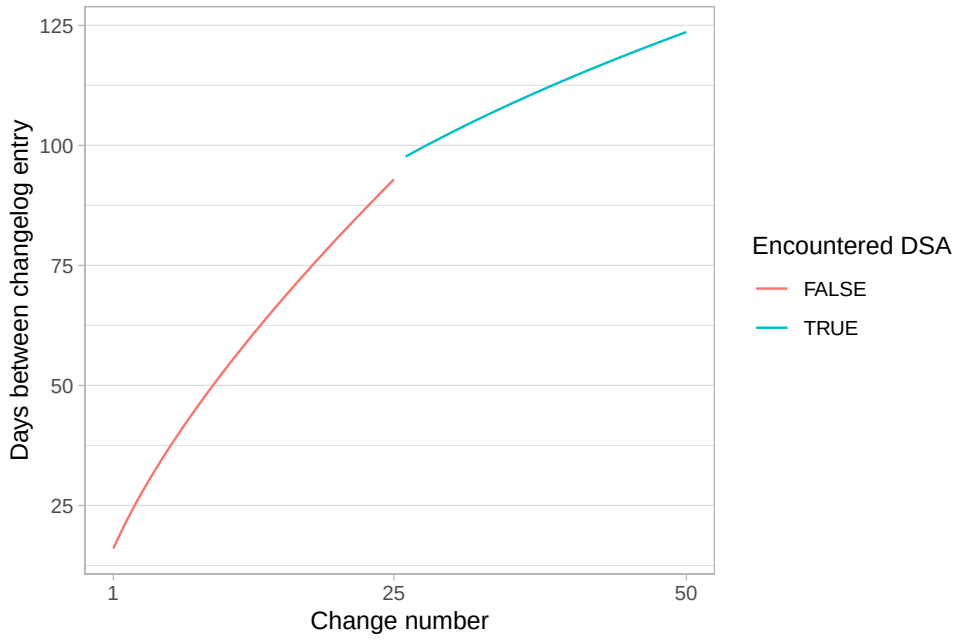

Figure 3: A prototypical project with 50 changelog entries, 25 before seeing a DSA, 25 after, with all other variables being held constant. Each integer along the X-axis represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`. Coordinates for the artificial project were predicted by Model (2), making use of *ruby-jquery-rails*, the project with the median random effects for the `package_name`.

affected by a DSA, afterwards, the rate of these code commits is increasing by about 1.48 days. That is to say, projects which are protective licensing generally demonstrate a resilience of 25.01%, whereas permissively licensed ones usually show a resilience of 37.20%, a difference of 12.19% (β =0.18, β 95%CI =0.09 – 0.26, p = 0.042). As was the case with the H1 results, I limit my interpretation to the estimate which is relevant to my hypothesis, the three-way interaction term between `change_number`, `seen_dsa`, and `is_protective`.

As with previous models, this is visually represented in Figure 4 on the following page where two prototypical projects, with 50 changelog entries each, differing only on license family, have been generated. Each experiences 25 software updates before encountering a DSA, whereafter they produce 25 further

changes. After predicting and plotting a value for each code update of each project with Model (3), a line representing the average development activity is placed atop the changes, stratified as before or after seeing the DSA, for each project. As was explained in the H2 results, I am interested in the difference of slopes, rather than any of the intercepts. As this model demonstrates, holding all other factors constant, the protectively licensed project, while reaffirming our findings described in the H1 results, experiences a smaller change post-DSA than the permissively licensed one, thus failing to support H3.

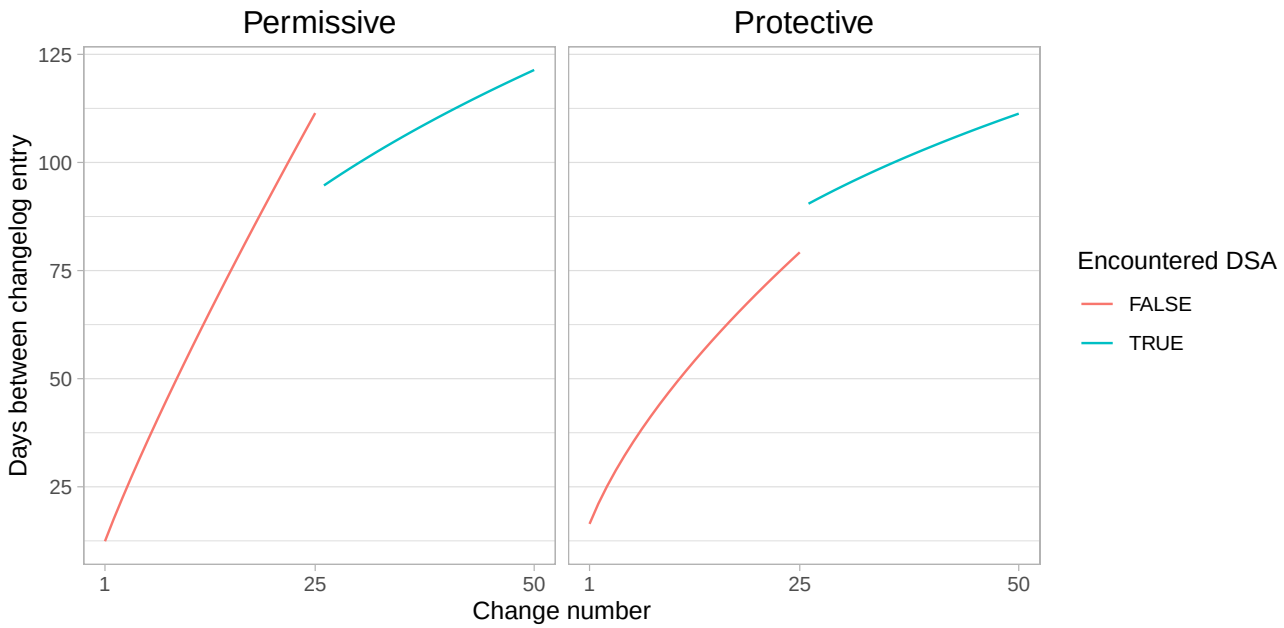

Figure 4: Two prototypical projects with 50 changelog entries each, differing by their license family (Left: Permissive, Right: Protective). Aside from that, all variables are held constant except for `seen_dsa` which is 0 for the first 25 entries, and 1 for the final 25, of each project. Each integer along the X-axies represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`. Coordinates for the artificial projects were predicted by Model (3), making use of *connectome-workbench*, the project with the median random effects for the `package_name`.

Discussion

The goal of this study was to explore the links between FLOSS development activity, founder decisions, crisis events, and resilience. To that end, I developed and tested three hypotheses; two focusing on the direct relationship to development activity, one on whether an interaction modified that relationship. H1 has mixed support, it shows that founder decisions have an impact on development activity. It also suggests that the difference between protective and permissive licensing becomes more pronounced as a project matures, and that protectively licensed projects fair better. H2 is supported, demonstrating the development resilience of FLOSS projects after a crisis event. H3 is not supported, suggesting that the echos of this particular founder's decision may not outweigh the effects of today's crisis event when it comes to development resilience.

Empirical Contribution

Broadly speaking, my research expands our understanding of FLOSS development, the archetype of commons based peer-production (Benkler & Nissenbaum, 2006). We now know more about its defining characteristic, the software license (Lessig, 2006). While the H1 results on page 20 generally align with the previous findings of Colazo and Fang (2009) and Stewart et al. (2006), opposing those of Fershtman and Gandal (2007) and Subramaniam et al. (2009), the story is not quite that clear cut. Instead, I find that the development activity of projects generally follows a declining pattern. However, protectively licensed FLOSS projects dwindle at a slower rate than permissively licensed ones. Another interesting finding given my hypothesis H1 on page 8, is that the first update of a protective protect takes longer than it would for a similar project that was permissively licensed. This was unexpected and may be a site of further research.

My research also expands our knowledge of an understudied, but important area, the impact of crisis events on FLOSS, through the lens of development resilience. As crisis events, "cannot be predicted but are expected" (Coombs & Holladay, 2010), these findings may realistically be used to help inform a future response. And aside from Raja and Tretter (2012) identifying resilience as one of, "three different attributes of project viability," I was unable to find scholars who had specifically studied this dimension.

As outlined in the H2 results on page 22, FLOSS shows considerate development resilience, with the rate of development activity speeding up by 26.99% on average following a crisis event.

These findings are statistically robust and supersede portions of the prior research for a variety of reasons. Firstly, the FLOSS landscape has changed in the past decade, and most of the seminal studies took place prior to even that. Not only is the data more recent, but it also spans a wider period, generally a project's entire lifespan, given that my earliest collected changelog entry occurred on January 18th 1995. Lastly, the scale of my study should be considered, given that it included 28,775 packages and 534,473 total observations, over double that of Subramaniam et al. (2009), and multiple orders of magnitude more than the rest.

Theoretical Contribution

Conceptually speaking, two ideas were central to my study, development resilience and founder decisions. Regarding resilience, I answer the call from Raja and Tretter (2012), which will hopefully lead to a resurgence of interest in this important topic. Additionally, I attempt to crystallize the relationship between crisis events and resilience, utilizing the communicative lens outlined by Coombs and Holladay (2010). By furthering the understanding of what development resilience entails, it will be easier to identify and study in the future.

Following an intuitive definition of constitutional decisions, I scoured the literature for some reference to its application in terms of software projects or community founders. Despite this, the only reference I was able to turn up happened to be from a seminal thinker in the discussion of communal public goods, Ostrom. Her definition was brief, but lent credence to the axes which I thought applicable, their broadness of impact and resistance to change (Ostrom, 1968). My extension comes in the form of a third affordance (Treem & Leonardi, 2013), the carrying of a founder's imprint.

Unfortunately, the H3 results were not in-line with the relationship described in my hypothesis H3 on page 10. This relationship, if shown, would have lent strength to my theory of founder decisions. That said, it is possible that the relationship between this particular crisis event and this particular founder decision, do not mirror other combinations. For instance, software forks and maintainer transitions are other examples of suitable crisis events, as is the selection of a programming language as a founder decision.

Implications for FLOSS Practitioners

As founders envision new FLOSS, and existing projects watch their development activity slow, these results provide useful insights. While my findings surrounding license choice were mixed, they do point towards the importance of founders being conscious of the impact their decisions have. The results also suggest that if the FLOSS project is being designed with a long-term community in mind, then a protective license might be preferred to a permissive one as the initial difference in development activity is made up for by the decreased rate of decay.

Additionally, a project might consider that the time and energy spent preparing for crisis events may be better focused elsewhere. Similarly, if a crisis event does happen, it is valuable to know that there will be increased development activity so that it does not become relied upon. Additionally, through awareness, this extra developer attention may be harnessed and directed towards new features, or nurtured in an attempt to cultivate long term supporters.

Finally, while resilience was not increased through the selection of a protective license, the H3 results on page 23 indicate that this is due to protectively licensed FLOSS generally having more rapid development activity. Given this, the Debian Project, and similar groups, should continue to push for the use of copyleft terms.

Potential Limitations

The validity of these findings may be affected by a number of threats and limitations. First, my operationalization of development activity, while utilized in one of the studies informing this one (Colazo & Fang, 2009), is very coarse. Subramaniam et al. (2009) point out that, "three measures of project success are inter-related," thus, despite the use of a varying-intercept model to control for unobserved heterogeneity, these interrelated factors may impact my results.

Additionally, my reliance on the ifrOSS license family schema may have come at the cost of an important distinction between individual licenses. What's more, by combining the strict and limited variants of the protective or permissive clauses, I may have covered over nuance which could be an important motivator driving the ideological license selection process.

Finally, the use of Debian changelogs means that I am looking at development activity filtered through a package maintainer, rather than measuring individual developers. As above, this choice may have obscured important nuance in the variability of development activity.

Future Research

There are a number of future areas of study to consider. First and foremost, the cross validation of these findings in other GNU/Linux distributions, both derivatives such as Ubuntu, and similarly sized competitors such as Fedora; and on other large-scale, software stores with publicly available longitudinal data, such as GitLab, would be valuable.

Second, as mentioned in the last paragraph of the Theoretical Contribution section, there are alternate combinations of founder decisions and crisis events which could be explored. Future research should be directed towards compiling the requisite characteristics to be categorized as one of these, gathering a list of features which fall into this taxonomy, and testing the different interactions among each.

Finally, there are always trade-offs when selecting a statistical methodology. Unidentified variables may have influenced these results and the analysis should be reproduced, replicated, and enhanced by other researchers.

Conclusion

By conceptualizing the founder decision and placing it in conversation with development resilience, I have identified and elucidated an important and understudied gap within the existing literature. My empirical results complement and extend the discussion surrounding the impact of FLOSS licensing on development activity. And, while not all of my hypotheses were fully supported, this just helps to highlight the many avenues for future research on the path towards greater understanding of the phenomena that is Free/Libre/Open Source Software.

ONLINE SUPPLEMENT

Data, code, and supplemental information for this thesis are available in the Harvard Dataverse at the following URL: https://doi.org/10.7910/DVN/IOE8DE

LIST OF FIGURES

1 The conceptual model grounding H1 to H3 on pages 8–10. It demonstrates the interactions between *development activity*, *project age*, *founder decisions*, and *crisis events*. Package age and founder decisions both have a direct effect on all development activity. Crisis events impact development activity after they occur. Founder decisions modify the relationship between crisis events and development activity. 11

2 Two prototypical projects with 50 changelog entries each, differing only by their license family (Left: Permissive, Right: Protective). Each integer along the X-axis represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`). Coordinates for the artificial projects were predicted by Model 1, making use of *rust-string*, the project with the median random effects for the `package_name`. 23

3 A prototypical project with 50 changelog entries, 25 before seeing a DSA, 25 after, with all other variables being held constant. Each integer along the X-axis represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`. Coordinates for the artificial project were predicted by Model (2), making use of *ruby-jquery-rails*, the project with the median random effects for the `package_name`. 24

4 Two prototypical projects with 50 changelog entries each, differing by their license family (Left: Permissive, Right: Protective). Aside from that, all variables are held constant except for `seen_dsa` which is 0 for the first 25 entries, and 1 for the final 25, of each project. Each integer along the X-axies represents a code update, where the X-coordinate is the `change_number`. The Y-coordinate is the number of days that occurred since the previous entry (i.e. `change_delta_days`. Coordinates for the artificial projects were predicted by Model (3), making use of *connectome-workbench*, the project with the median random effects for the `package_name`. 25

LIST OF TABLES

1 "Open Source Licenses" as categorized by the Institut für Rechtsfragen der Freien und Open Source Software (ifrOSS). I have utilized my own more consistent naming schema while including the original English translation in parentheses under each description. Full listing available at the ifrOSS License Center, https://ifross.org/en/license-center . 15

2 Mapping of concepts to operationalized variables of interest as found in the Debian Project data. `change_delta_days` is the dependent variable for all three linear mixed-effect models 1 to 3 on pages 19–20. 17

3 Linear mixed-model regression summaries of Models 1 to 3 on pages 19–20. 21

BIBLIOGRAPHY

Androutsellis-Theotokis, S., Spinellis, D., Kechagia, M., & Gousios, G. (2011). Open Source Software: A Survey from 10,000 Feet. *Foundations and Trends® in Technology, Information and Operations Management*, *4*(3–4), 187–347. https://doi.org/10.1561/0200000026

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, *67*(1), 1–48. https://doi.org/10.18637/jss.v067.i01

Benkler, Y. (2006). *The wealth of networks: How social production transforms markets and freedom*. Yale University Press.

Benkler, Y., & Nissenbaum, H. (2006). Commons-based Peer Production and Virtue. *Journal of Political Philosophy*, *14*(4), 394–419. https://doi.org/10.1111/j.1467-9760.2006.00235.x

Benkler, Y., Shaw, A., & Hill, B. M. (2015). Peer production: A form of collective intelligence. In T. W. Malone & M. S. Bernstein (Eds.), *Handbook of Collective Intelligence* (pp. 175–204). MIT Press. https://mako.cc/academic/benkler_shaw_hill-peer_production_ci.pdf

Bessen, J. (2005). *Open Source Software: Free Provision Of Complex Public Goods* (SSRN Scholarly Paper No. ID 588763). Social Science Research Network. Rochester, NY. Retrieved April 26, 2013, from https://doi.org/10.2139/ssrn.588763

Bonaccorsi, A., & Rossi Lamastra, C. (2003, September 9). *Licensing schemes in the production and distribution of open source software: An empirical investigation* (SSRN Scholarly Paper No. ID 432641). Social Science Research Network. Rochester, NY. https://doi.org/10.2139/ssrn.432641

Butler, B. S., Bateman, P. J., Gray, P. H., & Diamant, E. I. (2014). An attraction-selection-attrition theory of online community size and resilience. *MIS Quarterly*, *38*(3), 699–728. https://doi.org/10.25300/MISQ/2014/38.3.04

Champion, K., & Hill, B. M. (2021). Underproduction: An approach for measuring risk in open source software. *IEEE International Conference on Software Analysis, Evolution and Reengineering*. https://arxiv.org/abs/2103.00352

Chen, K., & Wagner, D. (2007). Large-scale analysis of format string vulnerabilities in Debian Linux. *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security - PLAS '07*, 75. https://doi.org/10.1145/1255329.1255344

Claes, M., Mens, T., Di Cosmo, R., & Vouillon, J. (2015). A Historical Analysis of Debian Package Incompatibilities. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 212–223. https://doi.org/10.1109/MSR.2015.27

Colazo, J., & Fang, Y. (2009). Impact of license choice on Open Source Software development activity. *Journal of the American Society for Information Science and Technology*, *60*(5), 997–1011. https://doi.org/10.1002/asi.21039

Coleman, E. G. (2005). Three ethical moments in Debian. *Social Science Research Network Electronic Journal*, 77. https://doi.org/10.2139/ssrn.805287

Coleman, E. G., & Hill, B. M. (2004). The social production of ethics in Debian and free software communities: Anthropological lessons for vocational ethics. In S. Koch (Ed.), *Free/Open Source Software Development*. Idea Group Inc. (IGI). https://mako.cc/academic/coleman_hill-social_production.pdf

Coombs, W. T., & Holladay, S. J. (Eds.). (2010, January 19). *The handbook of crisis communication* (1st ed.). Wiley-Blackwell. https://doi.org/10.1002/9781444314885

Crowston, K., Annabi, H., & Howison, J. (2003). Defining Open Source Software Project Success. *ICIS 2003 Proceedings*, 327–340.

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2008). Free/Libre open-source software development: What we know and what we do not know. *ACM Computing Surveys*, *44*(2), 1–35. https://doi.org/10.1145/2089125.2089127

de Laat, P. B. (2007). Governance of open source software: State of the art. *Journal of Management & Governance*, *11*(2), 165–177. https://doi.org/10.1007/s10997-007-9022-9

Deshpande, A., & Riehle, D. (2008). The Total Growth of Open Source. In B. Russo, E. Damiani, S. Hissam, B. Lundell, & G. Succi (Eds.), *Open Source Development, Communities and Quality* (pp. 197–209). Springer US. https://doi.org/10.1007/978-0-387-09684-1_16

Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., & Halderman, J. A. (2014). The Matter of Heartbleed. *Proceedings of the 2014 Conference on Internet Measurement Conference*, 475–488. https://doi.org/10.1145/2663716.2663755

Evans, S. K., Pearce, K. E., Vitak, J., & Treem, J. W. (2016). *Explicating Affordances: A Conceptual Framework for Understanding Affordances in Communication Research.* Retrieved November 1, 2018, from https://nls.ldls.org.uk/welcome.html?ark:/81055/vdc_100041775605.0x00002e OCLC: 1051956366

Fershtman, C., & Gandal, N. (2007). Open source software: Motivation and restrictive licensing. *International Economics and Economic Policy*, *4*(2), 209–225. https://doi.org/10.1007/s10368-007-0086-4

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587–598. https://doi.org/10.2307/25148740

Fogel, K. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media.

Foote, J., & Contractor, N. (2018). The behavior and network position of peer production founders. In G. Chowdhury, J. McLeod, V. Gillet, & P. Willett (Eds.), *iConference 2018: Transforming Digital Worlds* (pp. 99–106). Springer. https://doi.org/10.1007/978-3-319-78105-1_12

Fulk, J., Flanagin, A. J., Kalman, M. E., Monge, P. R., & Ryan, T. (1996). Connective and communal public goods in interactive communication systems. *Communication Theory*, *6*(1), 60–87. https://doi.org/10.1111/j.1468-2885.1996.tb00120.x

Galindo Duarte, J. A., Benavides Cuevas, D. F., & Segura Rueda, S. (2010). Debian Packages Repositories as Software Product Line Models. Towards Automated Analysis. *First International Workshop on Automated Configuration and Tailoring of Applications*.

Gamalielsson, J., & Lundell, B. (2017). On licensing and other conditions for contributing to widely used open source projects: An exploratory analysis. *Proceedings of the 13th International Symposium on Open Collaboration*, 1–14. https://doi.org/10.1145/3125433.3125456

Gasparini, M., Clarisó, R., Brambilla, M., & Cabot, J. (2020). Participation Inequality and the 90-9-1 Principle in Open Source. *Proceedings of the 16th International Symposium on Open Collaboration*, 1–7. https://doi.org/10.1145/3412569.3412582

Gelman, A., & Hill, J. (2007). *Data analysis using regression and multilevel / hierarchical models*. Cambridge University Press.

German, D. M., & Hassan, A. E. (2009). License integration patterns: Addressing license mismatches in component-based development. *2009 IEEE 31st International Conference on Software Engineering*, 188–198. https://doi.org/10.1109/ICSE.2009.5070520

German, D. M., Manabe, Y., & Inoue, K. (2010). A sentence-matching method for automatic license identification of source code files. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 437–446. https://doi.org/10.1145/1858996.1859088

Ghosh, R. A. (1998). Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet. *First Monday*, *3*(3). https://doi.org/10.5210/fm.v3i2.580

Ghosh, R. A. (2005). Understanding Free Software Developers: Understandings from the FLOSS Study. In J. Feller, B. Fitzgerald, S. A. Hissam, & K. R. Lakhani (Eds.), *Perspectives on Free and Open Source Software* (pp. 23–36). MIT Press. https://doi.org/10.7551/mitpress/5326.003.0006

Gibson, J. J. (1966). *The senses considered as perceptual systems* (Vol. 2). George Allen & Unwin Ltd.

Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, *6*(3), 25–39.

Herraiz, I., Shihab, E., Nguyen, T. H., & Hassan, A. E. (2011). Impact of Installation Counts on Perceived Quality: A Case Study on Debian. *2011 18th Working Conference on Reverse Engineering*, 219–228. https://doi.org/10.1109/WCRE.2011.34

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: An Internet-based survey of contributors to the Linux kernel. *Research Policy*, *32*(7), 1159–1177. https://doi.org/10.1016/s0048-7333(03)00047-7

Jensen, C., & Scacchi, W. (2011). License Update and Migration Processes in Open Source Software Projects. In S. A. Hissam, B. Russo, M. G. de Mendonça Neto, & F. Kon (Eds.), *Open Source Systems: Grounding Research* (pp. 177–195). Springer. https://doi.org/10.1007/978-3-642-24418-6_12

Kapitsaki, G. M., Tselikas, N. D., & Foukarakis, I. E. (2015). An insight into license tools for open source software systems. *Journal of Systems and Software*, *102*, 72–87. https://doi.org/10.1016/j.jss.2014.12.050

Kogut, B., & Metiu, A. (2001). Open-Source Software Development and Distributed Innovation. *Oxford Review of Economic Policy*, *17*(2), 248–264. https://doi.org/10.1093/oxrep/17.2.248

Krafft, M. F., Stol, K.-J., & Fitzgerald, B. (2016). How do free/open source developers pick their tools? a Delphi study of the Debian project. *Proceedings of the 38th International Conference on Software Engineering Companion*, 232–241. https://doi.org/10.1145/2889160.2889248

Kraut, R. E., & Fiore, A. T. (2014). The Role of Founders in Building Online Groups. *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 722–732. https://doi.org/10.1145/2531602.2531648

Kraut, R. E., Resnick, P., & Kiesler, S. (2012). *Building successful online communities: Evidence-based social design*. MIT Press.

Krishnamurthy, S. (2005). Cave or community? An empirical examination of 100 mature open source projects. *First Monday*, *7*(6). https://doi.org/10.5210/fm.v0i0.1477
(originally published in Volume 7, Number 6, June 2002)

Lakhani, K. R., & Wolf, B. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller, B. Fitzgerald, S. A. Hissam, & K. R. Lakhani (Eds.), *Perspectives on Free and Open Source Software* (pp. 3–22). MIT Press.

Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L., & Morasca, S. (2020). Open Source Software Evaluation, Selection, and Adoption: A Systematic Literature Review. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 437–444. https://doi.org/10.1109/SEAA51224.2020.00076

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, *50*(2), 197–234. https://doi.org/10.1111/1467-6451.00174

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. *The Journal of Law, Economics, and Organization*, *21*(1), 20–56. https://doi.org/10.1093/jleo/ewi002

Lessig, L. (2006). *Code: And other laws of cyberspace, version 2.0* (Version 2.0.). Basic Books. https://www.codev2.cc

Li, X., Moreschini, S., Zhang, Z., & Taibi, D. (2021). Exploring Factors and Measures to Select Open Source Software. *arXiv e-prints*, *2102*, arXiv:2102.09977. Retrieved March 30, 2021, from http://adsabs.harvard.edu/abs/2021arXiv210209977L

Mateos-Garcia, J., & Steinmueller, W. E. (2008). The institutions of open source software: Examining the Debian community. *Information Economics and Policy*, *20*(4), 333–344. https://doi.org/10.1016/j.infoecopol.2008.06.001

Meretz, S. (2007, April 10). *GNU/Linux is not a thing of value – and that is fine!* Retrieved February 21, 2021, from https://www.opentheory.org/linux-worthless/

Merten, S. (2004, December 12). *Gnu/Linux - Milestone on the Way to the GPL-society*. Stuttgart, Germany. Retrieved February 21, 2021, from https://www.opentheory.org/gplsociety/

Michlmayr, M., & Senyard, A. (2006). A Statistical Analysis of Defects in Debian and Strategies for Improving Quality in Free Software Projects. *The Economics of Open Source Software Development* (pp. 131–148). Elsevier. Retrieved February 4, 2020, from https://linkinghub.elsevier.com/retrieve/pii/B9780444527691500068

Nguyen, R., & Holt, R. (2012). Life and death of software packages: An evolutionary study of Debian. *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, 192–204.

Nussbaum, L., & Zacchiroli, S. (2010). The Ultimate Debian Database: Consolidating bazaar metadata for Quality Assurance and data mining. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 52–61. https://doi.org/10.1109/MSR.2010.5463277

O'Mahony, S., & Ferraro, F. (2007). The emergence of governance in an open source community. *Academy of Management Journal*, *50*(5), 1079–1106.

Ostrom, E. (1968). Constitutional decision-making: A logic for the organization of collective enterprises. *1968 Annual Meeting of the Midwest Political Science Association*, 28. https://dlc.dlib.indiana.edu/dlc/handle/10535/183

Ostrom, E. (1990). *Governing the commons: The evolution of institutions for collective action*. Cambridge University Press.

Project, D. (2004, April 26). *Debian Social Contract 1.1*. Retrieved February 11, 2021, from https://www.debian.org/social_contract

Raja, U., & Tretter, M. J. (2012). Defining and Evaluating a Measure of Open Source Project Survivability. *IEEE Transactions on Software Engineering*, *38*(1), 163–174. https://doi.org/10.1109/TSE.2011.39

Raymond, E. S. (1999). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary* (T. O'Reilly, Ed.). O'Reilly and Associates. http://www.catb.org/~esr/writings/cathedral-bazaar/

Robles, G., Gonzalez-Barahona, J. M., & Michlmayr, M. (2005). Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian. *Proceedings of the First Internation Conference on Open Source Systems*, 100–107.

Romer, P. M. (1990). Endogenous Technological Change. *Journal of Political Economy*, *98*, S71–S102. https://doi.org/10.1086/261725

Sadowski, B. M., Sadowski-Rasters, G., & Duysters, G. (2008). Transition of governance in a mature open software source community: Evidence from the Debian case. *Information Economics and Policy*, *20*(4), 323–332. https://doi.org/10.1016/j.infoecopol.2008.05.001

Sánchez, V. R., Ayuso, P. N., Galindo, J. A., & Benavides, D. (2020). Open Source Adoption Factors—A Systematic Literature Review. *IEEE Access*, *8*, 94594–94609. https://doi.org/10.1109/ACCESS.2020.2993248

Schweik, C. M., & English, R. C. (2012). *Internet success: A study of open-source software commons*. MIT Press.

Sen, R., Subramaniam, C., & Nelson, M. L. (2008). Determinants of the choice of open source software license. *Journal of Management Information Systems*, *25*(3), 207–240. https://doi.org/10.2753/MIS0742-1222250306

Sen, R., Subramaniam, C., & Nelson, M. L. (2011). Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? *Decision Support Systems*, *52*(1), 199–206. https://doi.org/10.1016/j.dss.2011.07.004

Spaeth, S., Stuermer, M., Haefliger, S., & von Krogh, G. (2007). Sampling in Open Source Software Development: The Case for Using the Debian GNU/Linux Distribution. *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. https://doi.org/10.1109/HICSS.2007.471

Stallman, R. M. (1984). *The GNU manifesto*. Retrieved September 21, 2013, from https://www.gnu.org/gnu/manifesto.html

Stallman, R. M. (2002, October). *Free software, free society: Selected essays of Richard M. Stallman*. GNU Press.

Stallman, R. M. (2009). Why "Open Source" misses the point of free software. *Communications of the ACM*, *52*(6), 31–33. https://doi.org/10.1145/1516046.1516058

Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects. *Information Systems Research*, *17*(2), 126–144. https://doi.org/10.1287/isre.1060.0082

Stewart, K. J., & Gosain, S. (2006). The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly*, *30*(2), 291–314. https://doi.org/10.2307/25148732

Subramaniam, C., Sen, R., & Nelson, M. L. (2009). Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, *46*(2), 576–585. https://doi.org/10.1016/j.dss.2008.10.005

Team, R. C. (2021). *R: A language and environment for statistical computing*. Vienna, Austria. https://www.R-project.org/

The pandas development team. (2021, February). *Pandas-dev/pandas: Pandas* (Version latest). https://doi.org/10.5281/zenodo.3509134

Treem, J. W., & Leonardi, P. M. (2013). Social media use in organizations: Exploring the affordances of visibility, editability, persistence, and association. *Annals of the International Communication Association*, *36*(1), 143–189. https://doi.org/10.1080/23808985.2013.11679130

Viseur, R., & Robles, G. (2015). First Results About Motivation and Impact of License Changes in Open Source Projects. In E. Damiani, F. Frati, D. Riehle, & A. I. Wasserman (Eds.), *Open Source Systems: Adoption and Impact* (pp. 137–145). Springer International Publishing.

von Hippel, E., & von Krogh, G. (2003). Open source software and the 'private-collective' innovation model: Issues for organization science. *Organization Science*, *14*(2), 209–223. https://doi.org/10.1287/orsc.14.2.209.14992

von Krogh, G. (2003). Open-Source Software Development. *MIT Sloan Management Review*, *Spring*. Retrieved April 26, 2013, from http://sloanreview.mit.edu/article/opensource-software-development/

Walden, J. (2020). The Impact of a Major Security Event on an Open Source Project: The Case of OpenSSL. *Proceedings of the 17th International Conference on Mining Software Repositories*, 409–419. https://doi.org/10.1145/3379597.3387465

Weber, S. (2004). *The success of open source*. Harvard University Press. https://www.hup.harvard.edu/catalog.php?isbn=9780674018587

Wes McKinney. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag. https://ggplot2.tidyverse.org

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Yilek, S., Rescorla, E., Shacham, H., Enright, B., & Savage, S. (2009). When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 15–27. https://doi.org/10.1145/1644893.1644896

Yunwen Ye, & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. *25th International Conference on Software Engineering, 2003. Proceedings.*, 419–429. https://doi.org/10.1109/ICSE.2003.1201220

Zacchiroli, S. (2015). The Debsources Dataset: Two Decades of Debian Source Code Metadata. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 466–469. https://doi.org/10.1109/MSR.2015.65

Zimmermann, T., & Casanueva Artís, A. (2019). Impact of switching bug trackers: A case study on a medium-sized open source project. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 13–23. https://doi.org/10.1109/ICSME.2019.00011

ACKNOWLEDGMENTS

Throughout the writing of this thesis, I received a great deal of support and assistance from a vast number of allies. While all may not be named here, please know that I would could not have accomplished this without you.

First, I would like to thank my advisor, Mako, who has now assisted me through two graduations at the University of Washington. You are an inspiration, both in the academy and FLOSS community at large. Please know that if I were to include you in all of the follow acknowledgments that you were a part of, this thesis would likely need a different title. My mind and pathway have both been expanded through our work together. Thank you.

I would also like to highlight and thank Kaylea Champion, my cohort companion, desktop neighbor, and consistent collaborator. You have always been there to reflect and refract ideas, honing them along the way during these arduous years. While it goes without saying, you truly are a champion. Thank you.

This last push to completion would likely not have been possible without the help from Dr. Kirsten Foot. Thank you for your structure and organizational advice. I am incredibly grateful to have you on my committee.

I would like to acknowledge all of my colleagues in the Community Data Science Collective. You were few when I joined and have grown extensively, but the core interests and commitment to sound inquiry within the social sciences has been an invaluable anchor.

Thanks goes out to those of you who allowed me to lend a hand to your work as a co-author. Thank you Sayamindu Dasgupta, Andrés Monroy-Hernández, Sneha Narayan, Nathan TeBlunthuis, and Aaron Shaw.

To those CDSC members who have been here nearly my entire journey, thank you Jeremy Foote, Emilia Gan, Jim Maddock, and Charles Kiene. And to the CDSC members who have brought such brightness in these past turbulent years, thank you Sejal Khatri, Regina Cheng, Carl Colglazier, Stefania

Druga, Floor Fiers, and Sohyeon Hwang. Additionally, I must thank two proto-CDSC members for planting the seeds which blossomed into this stage of my academic career, thank you Samantha Shorey and Sam Woolley.

Beyond my research group, I would like to extend thanks to many of my colleagues at the University of Washington. Countless hours of writing were shared silently, previous in person, and lately on screen, alongside Dharma Dailey and Amirah Majid, thank you both. To my crafting partner in crime, thank you Kathryn Shoryer for your creativity reverberation.

I would like to extend my thanks to everyone in the Department of Communication, while also highlighting a few names of those who I've spent more time with, and/or received exceptional support from, over these past years. Thank you to all of the faculty, especially Caley Cook, Anita Verna Crofts, Ralina L. Joseph, Matt McGarrity, LeiLani Nishime, Katy Pearce, Adrienne Russell, and Anissa Tanweer; all of the staff, especially Troy Bonnes, Winnie Cao, Wendy Durant, Tommy Ferguson, Anu Hastings, Megan Jeffrey, Nick Myers, Edith Olguin, Virgel Paule, Erika Samson, and Heather N. Werckle; and all of my fellow graduate students, including Benjamin Compton, Meagan Doll, Marcus Johnson, Erin Keoppen, Kyle Kubler, KC Lynch, Carolina Nieto, Madison Snider, Polly Straub-Cook, Meshell Sturgis, Anna Lee Swan, Rian Wanstreet, and Joseph Whitt.

To the weekly discussants and participants who I've joined over the past four years, I know that I am missing many of your names but your faces and memories are well ingrained: David W. McDonald, Morten Warncke-Wang, Mark Zachry, Sage Ross, Taryn Bipat, Ridley Jones, Kai Lukoff, Amanda Menking, and Himanshu Zade in the Social Computing Reading Group; Ryan Calo, Batya Friedman, Yoshi Kohno, Hannah Almeter, Maria P. Angel, Alex Bolton, Stephanie Ballard, Cameron Cantrell, Ivan Evtimov, Bernease Herman, Karl Koscher, Lassana Magassa, Savannah McKinnon, Katherine Pratt, Peter Ney, Espen Scheuer, Anna Kornfeld Simpson, Leron Vandsburger, and Meg Young with the Tech Policy Lab; and everyone who has been involved with DUB, thank you all.

I would also like to thank all of the members at the Washington Yacht Club, especially my Commodore crew, for these many years of support. Thank you Maximilian Chmielinksi, Angie Cuadra-Palacios, Tanner Graves, Kristan Lund, Milo Martin, Lauren Strand, and all of the other sailors.

Beyond academia, there are so many who I would like to thank, but as this must cease at some point, I

will begin by thanking Bryan Newbold, Mika Matsuzaki, Will Scott, and Lucy Lu Wang of ELS. I eagerly await our next community meal. Thank you Adam Monsen, Deb Nicholson, Chris Petersen, and Rob Smith for founding the SeaGL flock and those all of those who have volunteered their time to build and participate such a wonderful annual conference. Thank you adroit, alignwaivers, chreekat, iko, mray, msiep, smichel17, and wolftune for all of your support in our shared dream of clearing the path towards a free/libre/open future with Snowdrift.

Before closing, I would like to extend thanks to some of those who have stuck with me throughout this, and for some, much more. Thank you Craig and Jenny Hale, Andrew Kvalheim, James Raymond, Zach Zulch, Don Ankney, Josh Armour, Mark Atwood, Will Bailey, Brandon Barton, Molly de Blanc, VM Brasseur, Lee Colleton, John Davis, Morgan Davis, Mairi Dulaney, Emily Dunham, Eve Elle, Alex Floyd, Rachael Frost, Peaches Gall, Ian Gallagher, Morgan Gangwere, Joshua Gay, Chris Gilbert, Kasey Goodman, Chris Hacking, Mercedes Haefer, Alex Haines, Brian Harden, Bri Hatch, Rick Hauptman, Andy Heald, Joey Hess, Katharine Hough, David Hulton, Tim Huynh, Emily, Keith Herrington, Esther Jang, Ben Kero, Sean Leonard, Ben Lewis, John Jinneman, Mikey Jensen, Alex Jordan, Cameron King, Peaks Krafft, Matt Lavallee, Christopher and Morgan Lemmer-Webber, Jake Loukkula, Aimee Maree, Patrick Masson, Aaron McCloud, Gabrielle Mehlman, Josh Michaels, Amelia Min-Venditti, Jonathan Morgan, John Murphy, Merrielle Ondreicka, Aris Powell, Niki Frances Quinzel, Nick Rasheab, Morgan Redfield, Blaise Ritchie, Chris Rose, Jesse Skellington, Olivia Smith, William Song, John Sullivan, Cody Thompson, Zaq Wiedmann, Bill Wright, Stefano Zacchiroli, and Valorie Zimmerman for all of your support .

Finally, I would like to thank the rest of my family, my friends, my clubs, conferences, conventions, events, faires, and other odd gatherings of like-minded folk. You have all contributed to this work, through your inspiration and welcoming ways. Despite the constant advice given to ”just say no” whenever volunteering arose, I am proud to have helped in the co-creation of so many of these experiences. Thank you all so very much.

VITA

Salt is a graduate student with the Community Data Science Collective. He studies online peer production communities such as Linux, Scratch, and Wikipedia. Salt has a background in free/libre/open source software, information security, and community organizing.

Correspondence concerning this thesis should be addressed to:

Wm Salt Hale

PO Box 47042

Seattle, WA 98126

Website: https://www.altsalt.net

https://orcid.org/0000-0001-5329-7811