

2015

Engaging developers in open source software projects: harnessing social and technical data mining to improve software development

Patrick Eric Carlson
Iowa State University

Follow this and additional works at: <http://lib.dr.iastate.edu/etd>

 Part of the [Computer Engineering Commons](#), [Computer Sciences Commons](#), and the [Library and Information Science Commons](#)

Recommended Citation

Carlson, Patrick Eric, "Engaging developers in open source software projects: harnessing social and technical data mining to improve software development" (2015). *Graduate Theses and Dissertations*. 14663.
<http://lib.dr.iastate.edu/etd/14663>

This Dissertation is brought to you for free and open access by the Graduate College at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**Engaging developers in open source software projects: Harnessing social and
technical data mining to improve software development**

by

Patrick Eric Carlson

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Human-Computer Interaction

Program of Study Committee:

Judy M. Vance, Major Professor

Tien Nguyen

James Oliver

Jon Kelly

Stephen Gilbert

Iowa State University

Ames, Iowa

2015

Copyright © Patrick Eric Carlson, 2015. All rights reserved.

DEDICATION

This is dedicated to my parents who have always supported me.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
ACKNOWLEDGEMENTS	ix
ABSTRACT	x
CHAPTER 1. OVERVIEW	1
1.1 Introduction	1
CHAPTER 2. BACKGROUND	3
2.1 Open Source Software	3
2.1.1 History	4
2.1.2 Communities	8
2.1.3 Motivation	9
2.1.4 Tools	15
2.2 Social Technical Congruence (STC)	19
2.2.1 Classical Hierarchy	19
2.2.2 Distributed Structure	20
2.2.3 STC Algorithm Example	24
2.3 Socialization of New Developers	29
2.3.1 Climate and Culture	33
2.3.2 Relationships and Influence	35
2.3.3 Socializing New Members	36
2.3.4 Data Mining and Network Analysis	40

2.4	Current Problems in OSS	41
2.5	Existing STC Tools	43
2.6	Contributions	44
CHAPTER 3. SURVEY OF COMMUNITIES		46
3.1	Communities	46
3.2	Survey	47
3.3	Characteristics of Core Developers and New/Lurking Individuals	48
3.4	Survey Results	49
3.4.1	Overall Responses	52
3.4.2	Qualitative Results	60
3.4.3	Comparing New and Veteran Developers	65
3.5	Conclusion	70
CHAPTER 4. ALGORITHM AND WEBSITE DEVELOPMENT		74
4.1	Jamii Online Website	74
4.1.1	Formative Usability Testing	78
4.2	Impactful Commits Algorithm	80
4.3	Degree of Knowledge Algorithm	87
4.4	Communication Suggestions Algorithm	95
CHAPTER 5. WEBSITE ANALYSIS		105
5.1	Jamii Data	105
5.1.1	Website Analytics	109
5.2	Quantitative Survey Results	111
5.3	Qualitative Interview Results	111
CHAPTER 6. SUMMARY AND DISCUSSION		122
6.1	Algorithm Limitations and Assumptions	123
6.2	Future Work	124
6.3	Conclusion	126
APPENDIX A. COMMUNITY SURVEY		127

APPENDIX B. JAMII SURVEY	141
BIBLIOGRAPHY	159

LIST OF TABLES

		Page
Table 3.1	Measures and Associated Questionnaires	49
Table 3.2	Country Distribution	50
Table 3.3	Highest Education/Schooling	51
Table 3.4	Time Spent in Community Each Week	54
Table 3.5	Paid Compensation (Employees)	55
Table 3.6	Qualitative Comments Mozilla	61
Table 3.7	Qualitative Comments KDE	62
Table 3.8	Qualitative Comments Mozilla	63
Table 3.9	Qualitative Comments KDE	64
Table 3.10	Qualitative Comments Mozilla	65
Table 3.11	Qualitative Comments KDE	66
Table 3.12	Correlation Table for Skills/Characteristics for New Members	67
Table 3.13	Correlation Table for Member Joining and Socialization	68
Table 3.14	Correlation Table for Transactive Memory	70
Table 4.1	Revisions to the File	93
Table 4.2	Authorship Information for Source Code File	94
Table 5.1	Neo4j Database Statistics	107
Table 5.2	CLOC Mozilla Statistics	108
Table 5.3	CLOC KDE Statistics	109
Table 5.4	Total Analytics Results	110

LIST OF FIGURES

		Page
Figure 2.1	Onion Model of OSS (Figure 1 of Crowston and Howison (2005))	12
Figure 2.2	Communication Between Developers of Dependent Modules	23
Figure 2.3	Network Structure Between Developers and Source Code	24
Figure 3.1	Survey Development and Order	47
Figure 3.2	Spoken/Written Languages	51
Figure 3.3	Number of Years in Community	52
Figure 3.4	Experience Contributing to FLOSS	53
Figure 3.5	Computer Programming Experience	53
Figure 3.6	Integration Level: Current and Future	55
Figure 3.7	Delta Difference: Wanted Integration vs. Current Integration	56
Figure 3.8	New Members Join	57
Figure 3.9	New Member Integration	57
Figure 3.10	New Member Respect	58
Figure 3.11	Rating of Skills for New Members	59
Figure 3.12	Keep Track Processes	59
Figure 3.13	Transactive Memory	60
Figure 4.1	Data Mining Sources	74
Figure 4.2	Example Graph with Nodes and Edges	75
Figure 4.3	Neo4j Database Structure	77
Figure 4.4	Front Page of Jamii Website	79

Figure 4.5	Code Impact Section of Jamii Website	81
Figure 4.6	Code Impact Details	82
Figure 4.7	PageRank Graph Example	83
Figure 4.8	PageRank Graph Example With Values	85
Figure 4.9	Code Impact Example	86
Figure 4.10	Code Impact Example Page Rank Values	87
Figure 4.11	Developer Knowledge Section of Jamii Website	96
Figure 4.12	Developer Knowledge Details	97
Figure 4.13	Communication Suggestions of Jamii Website	98
Figure 4.14	Communication Suggestions Details	99
Figure 4.15	Communication Suggestions Initial Data	100
Figure 4.16	Consolidation of Knowledge and Communication Scores	101
Figure 4.17	Weighting File Scores	102
Figure 5.1	Number of Visits to the Jamii Website	110

ACKNOWLEDGEMENTS

My graduate education would not have been possible without the help and support of many people. First, I would like to thank my advisor, Judy Vance who has spent immeasurable time providing thoughtful advice and technical expertise. Her impact on my graduate experience will remain with me throughout my life. I would also like to thank my committee members, Tien Nguyen, James Oliver, Leigh Tesfatsion, Jon Kelly, and Stephen Gilbert who provided thoughtful advice and support on my research. The professors and staff of VRAC have been extremely supportive and have provided a wide variety of interesting classes and topics. Thank you to my undergraduate professors at Simpson College who started my interest in research and supported undergraduate research in general. Thanks to my fellow graduate students, Daniela, Denis, Isaac, Ryan, Tim, Meisha, and Leif for the crazy afternoon discussions. I would like to thank all my friends from the badminton club for all the late night dinners after practice. And finally, I would like to thank my parents who have supported me 100% throughout my childhood and as an adult. Your support and those of countless others have helped me immensely over these years and I truly appreciate your guidance.

ABSTRACT

As software development has evolved, an increasing amount of collaboration and management is done online. Open source software, in particular, has benefited greatly from communication and collaboration on the Internet. As software projects increase in size, the codebase complexity and required communication between developers increases. The barriers of entry for development participation are not only technical in nature but involve understanding the changing dynamics of the community.

Social Technical Congruence (STC) attempts to understand and model the synergies between technical development and communication. Motivated by this theory, three algorithms were developed that leverage data from version control history and email mailing list communication to help developers better understand the community. The “code impact” algorithm identifies commits that are impactful to the overall technical structure of the source code. The “developer knowledge” algorithm calculates the knowledge that a developer or developers have for a particular section of code. Lastly, the “communication suggestions” algorithm provides suggestions about who a developer could be increasing communication with on the mailing list based on shared technical dependencies and recent communication. These algorithms were implemented in an online website called “Jamii”. Through an interactive front-end, the website provides relevant development-centric information to the community. The website was evaluated with Mozilla and KDE, two large open source communities.

Quantitative and qualitative analysis provide details about the socialization process before the website was developed through an initial survey as well as afterwards. The results from the initial survey paint a primarily positive picture of socialization and inclusion for KDE and a more complex process of socialization and inclusion for Mozilla because of the large size of its community and codebase. In many cases for both communities, the ratings on a number of factors regarding the socialization process and effective coordination were actually higher for

the beginner participants as compared to the veterans. This suggests that new participants may have an overly-inflated positive view than veteran members. This work provides fundamental contributions to the field of STC and may generalize to other distributed online collaboration efforts.

CHAPTER 1. OVERVIEW

1.1 Introduction

Technology and global connectivity are driving economic growth throughout the world at an increasing pace. The perception of the world as a “global village” is, in part, a consequence of numerous technological advances in communication that have made the world feel smaller (McLuhan, 1964). One area that has adapted and thrived as a result of these advances is open source software (OSS). Based on the notion that a program’s source code should be available and accessible to anyone, OSS has benefited greatly from advances in electronic communications such as email and other text-based communications. As a result, developers from all parts of the globe can participate in OSS development. OSS enables the saving of millions of dollars for companies both small and large (Walli, Gynn, & Rotz, 2005) and the growth, in terms of software produced and projects initiated and completed, is increasing exponentially (Deshpande & Riehle, 2008). Of course, much like any other product, process or service, the efficiency and productivity is only as good as its program structure and the communication among its developers. The health of the project through retention of new contributors as well as individual advancement in the online community are critical components. Communication patterns of the individuals working on the design as well as dependencies between various sub-parts of the product being designed are intricately entwined and affect the overall quality.

The field of Social Technical Congruence (STC) (Cataldo, Wagstrom, Herbsleb, & Carley, 2006) attempts to understand and model the synergies between technical development and communication. OSS presents a unique test-bed to explore these synergies due to the social and technical aspects of participation in the OSS community, as well as the vast repositories of open data that are available. The overall community is incredibly diverse and combines

philosophical, cultural, and a wide variety of different skill sets and abilities with community participants supporting everything from privacy rights to low-level operating system design. STC is uniquely positioned to provide the necessary analysis in order to help new users join existing communities.

The following research explores the connections between the social and technical structures that exist in large communities such as OSS and the health of the community as measured using the core principles of STC. Algorithms have been created and tested through the use of an online website that provides information relevant to new software developers.

The work is divided into the following sections. Chapter two outlines the history of open source and the motivation for this work. Chapter three details the results of an initial survey of two large open source communities, Mozilla and KDE. The survey focuses on member perception of the socialization process for bringing in new members. Chapter four provides information about the development of an online website that parses and analyzes source code version control and email mailing list communication to provide recommendations and reasoning for developers. Three algorithms are provided as part of the tool. First, source code commits are processed to determine which commits have the highest change impact to the overall technical structure of the source code. Second, users can select a source code file or a portion of the file and calculate which developers have expertise in the file based on the recency and number of commits. Lastly, a communication suggestions algorithm gives recommendations for who a developer should be increasing communication with on the mailing list based on shared source code dependencies and the amount of current communication. Chapter five is the application of this online tool to the two open source communities including a final survey and interviews to determine their reaction and applicability for improving the socialization process. Chapter six provides a summary and conclusion.

This tool and algorithms more precisely and efficiently allow new and existing open source members the ability to understand the social structure and assist in the development process. This work provides fundamental contributions to the field of STC and may generalize to other distributed online collaboration efforts.

CHAPTER 2. BACKGROUND

2.1 Open Source Software

The OSS model has grown significantly over the years from the academic world of sharing small pieces of source code to large communities of developers creating full-fledged software systems with substantial corporate sponsorship (e.g., IBM Eclipse, Mozilla, Red Hat Linux, Ubuntu Linux, etc.). The main impetus behind the creation of the open source concept was the philosophy that the source code of a program should be accessible to everyone. This accessibility allows individuals to add features, fix bugs, and provide suggestions with the understanding that the changes made must be available to the entire community at large. The Open Source Initiative (OSI) defines open source as having among others the following properties ([Open Source Initiative, 2015](#)):

- “The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources.”
- “The program must include source code, and must allow distribution in source code as well as compiled form.”
- “The license must allow modifications and derived works...”
- “The license must not discriminate against any person or group of persons.”
- “The license must not place restrictions on other software that is distributed along with the licensed software.”

In this way, OSS evolves through the contributions and evolution of the developers as well as the users. A prominent benefit from this type of community is that it takes advantage of its size to create a quality product or service. The freedom resulting from an open code-base can result in significant commercial ([Spinellis & Giannikas, 2012](#)) and grassroots communities emerging around projects.

The core freedoms and principles of OSS transcend national borders. Open source software projects often involve people who are geographically diverse and who may have different languages and social customs. The modification and availability of the source code helps them communicate and collaborate. Creating an open source project and community involves not only technical skills but recruiting users and trusting them. Contributors not only provide new or different source code but also documentation, translations, bug reports, and other functions. Power and control in these communities emerges quickly. This can often lead to a high barrier of entry for a beginner developer or programmer looking to become a contributor to the project. Understanding the communication and society behind open source projects are important not only for examining the community as a whole but also for new users who wish to join.

2.1.1 History

Today, the widespread proliferation of computers and users is much different than it was in the 1960's and 1970's. At that time, personal computers did not exist. Computers were huge main frame systems that required large air conditioned and climate controlled rooms. Programs were created on 80 column punch cards and manually fed into the main frame system. The community of computer users was very small in comparison to the number of users today. Source code, the underlying instructions of the computer, was often freely shared throughout the community. Around the mid 1970's, the Unix operating system was beginning to gain use around the world ([Ceruzzi, 2003](#)). ARPANET, the Defense Advanced Projects Research Agency (DARPA) funded precursor to the Internet was gaining in popularity ([Leiner et al., 2009](#)). Academics was the area where these new emerging technologies were mostly being researched and developed. Academia has always been a fairly open community of sharing information and ideas due to the culture within higher education. In the United States in particular, the events

and protests of the Vietnam War primarily on college campuses clearly had an impact on the American psyche. This group who was heavily influenced by the culture and society of the time which supported freedom of speech and the “hippie” culture included the same individuals who advanced two new innovations, the Internet and the personal computer. These two technologies would be interlaced and heavily dependent on one another in subsequent years. Through the youth movement of the 1970’s and the culture that developed, these individuals then brought this ethos to their work environments.

Corporations began to realize the potential profit center revolving around the development and licensing of software. Protecting their intellectual property in the software through the use of copyrights became more important. This source code was often closed in that users who purchased a copy of the software were unable to see the underlying source code and could only run the program as it was provided. This closed source code issue presented a variety of problems for many users, particularly those in academia who wished to fix bugs or problems with the code or even add new features. In the early academic research days of computing, it was not uncommon to share code with others. At the MIT Media Lab, Richard Stallman would become, in essence, the “father” of the open source movement. After becoming increasingly frustrated with the inability to view the source code and understanding where this could lead technology in the future, he left his job at MIT in 1984 and started the Free Software Foundation (FSF) ([Richard Stallman, 2015a](#)). “Software for him was not just a tool to run computers. It ultimately was a manifestation of human creativity and expression. Even more importantly, software represented a key artifact of a community that existed to solve problems together for the common good. It was as much about the kind of society you lived in as the technology you used” (pg. 47) ([Weber, 2004](#)). The goal of this nonprofit organization was to create a free and open operating system that anyone could use, modify, and share ([Ceruzzi, 2003](#)). At the time given the popularity of the proprietary Unix operating system, this system would be technically modeled in a similar fashion. Created in 1983, the project was entitled GNU ([Richard Stallman, 2015b](#)) which is a recursive acronym called GNU’s Not Unix which was envisioned to be a free and open codebase and competitor to Unix ([Williams, 2002](#)). The GNU Manifesto was created to articulate the purpose and goals of the project, as well as clarify the difference between “free”

and “freedom”. As Stallman explains in a footnote, “I have learned to distinguish carefully between ‘free’ in the sense of freedom and ‘free’ in the sense of price. Free software is software that users have the freedom to distribute and change. Some users may obtain copies at no charge, while others pay to obtain copies and if the funds help support improving the software, so much the better. The important thing is that everyone who has a copy has the freedom to cooperate with others in using it” ([Stallman, 1985](#)). A famous quotation by Richard Stallman outlines this when he explains that, “you should think of ‘free’ as in ‘free speech’, not as in ‘free beer’” ([Free Software Foundation, 2015](#)).

Understanding the legal structure behind this, Stallman realized that users might take free software, add something new, and spin it off into a proprietary product. Then, users would not have these freedoms for this improved version. This led to the creation of the General Public License (GPL) or “copyleft” license. It uses copyright laws to protect the source code and make sure that derivative works from free software stay free.

“Software that is licensed under the GPL cannot be made proprietary. Derivative works from free software must also remain free. The GPL goes further. It does not allow the use of GPL’ed code in any proprietary implementation at all. It is not permitted under the GPL to combine a free program with a nonfree program unless the entire combination is then released as free software under the GPL” (pg. 48-49) ([Weber, 2004](#)).

In this way, if any user modifies a piece of GPL code, the license requires that any code they release also fall under the GPL. While never realizing the full potential of creating an entirely free operating system, the GNU project was monumental in shaping the open source community as a whole and created a wide variety of important projects still in use today. Everything from text editors, compilers, web-browsers, video editing software, and a multitude of other software is now available. The GPL license is utilized by a significant number of open source projects today.

In the early 1990’s in Helsinki Finland, Linus Torvalds started designing and implementing a new operating system ([Torvalds & Diamond, 2002](#)). Torvalds was a graduate student in

Computer Science at the University of Helsinki. He modeled his creation after an operating system called Minix which was a clone of Unix. The GNU tools that Richard Stallman was working hard to improve were used by Linus to program and compile the base Linux kernel. After developing the initial software, he released the code under the GPL license and solicited help from the Internet community at large. Suddenly, people all over the world were contributing bug fixes, code improvements, and new features. Unix was dying and Linux, as it was called, was fast replacing it as the defacto standard. Torvalds licensed Linux under the GPL which helped not only to protect it but spur development. It was at this point in time that OSS gained in popularity and received worldwide recognition. No longer just a fad used by technically savvy users, open source software was being used by companies and individuals all over the world. Linux distributions sprung up offering different software configurations and customization. One that particularly embodied the spirit of open source software was Debian. As of 2015, Debian is a long running and popular Linux Operating System. The Debian Social Contract was created to ensure that Debian remained open source and was accessible to anyone ([Debian, 2015](#)). Many Linux distributions have been spun off Debian including Ubuntu, one of the most popular distributions today. Open source software was transformed by these key individuals as well as the hundreds (if not thousands) of contributors around the world.

One area of debate and contention in the community is the terminology and philosophy espoused by supporters. Richard Stallman and others believe that the freedom granted to individuals through free software is the most important value. Popular terms such as “open source” are believed not to get at the heart of the issue which is the freedom of using technology. Richard Stallman in particular, makes a point to correct people when they say “open source” or “Linux” and not “GNU/Linux”. The term “open source” will be used in this paper.

From a business standpoint, the goal of developing proprietary software is to make a profit for the company. The bottom line is selling the software to users at a profit. This business model came out of traditional business models of creating, producing, and selling products. The fact that source code is not a physical product did not matter. The process was a top-down profit driven formula. However, for a business that does not rely on selling software, opening up the code provided some real benefits. The cost of developing a proprietary software system for

internal use may be higher than developing and managing an open source project. While not having direct control over the project may seem dangerous, since the code is always available to others, companies have the ability to fork projects if the direction of the community differs from their goals.

The main philosophy behind open source software is heavily tied to how the system works.

“The key element of the open source process, as an ideal type, is voluntary participation and voluntary selection of tasks. Anyone can join an open source project, and anyone can leave at any time... Each person is free to choose what he [or she] wishes to work on or to contribute. There is no consciously organized or enforced division of labor” (pg. 62) ([Weber, 2004](#)).

Given this environment, it is amazing that complicated software can be produced in the first place. Since there is no central authority making decisions on what goes into the software, the system essentially evolves over time. The communication and power structure that emerges in an open source community has a huge impact on shaping and creating the end product.

2.1.2 Communities

Differences in opinion and direction do occur in open source communities. If one person or group does not like the direction the project is going, they can “fork” the project and split it off into something separate and develop it the way they see fit. Forking a project refers to the process of taking the source code and branching off and creating a new project and community. The main benefit is that the source code is continuously available through all of the various iterations of a project.

Open source communities are dynamic systems of both people and technical source code. Developers often use version control software such as Git, Subversion, and CVS to keep track of source code changes. The version control system monitors all changes as well as the author of the change. Individual members are identified through their communication with other members as well as through their contributions of code. This perception of skill and value in the community takes place mostly through text-based communication. Becoming a senior

member of the community takes time, effective written communication skills, and technical expertise. The diversity of skill, social power differences, cultural differences between members, and other diversities has been shown to have both positive and negative impacts on open source communities ([Daniel, Agarwal, & Stewart, 2012](#)).

Online open source communities can be identified as cultures of participation and [Fischer \(2011\)](#) outlines a set of properties of these broad online communities. First, contributors must feel that they are capable of making changes to the system. In the open source realm, easy access to source code via version control as well as instructions for building the software and the necessary third-party libraries are important. Second, the benefits of the time and investment by the individual must be visible. This ties heavily into the motivation of why individuals participate in open source projects. Third, the online environment must support the tasks that individuals will work on. Fourth, barriers to sharing must be low in order to facilitate the sharing of the resulting change. And finally, that individuals must be open to change in the system and open to new people joining the community.

In OSS, connections such as community structure and power play a key role. Unlike traditional software development where there is a rigid leadership structure and programmers are assigned pieces of code to write, OSS relies heavily on volunteers. While companies do contribute money and programmers to OSS projects like the Linux kernel, many projects have only a handful of developers working in their spare time or for personal enjoyment. Since there is no requirement to participate in the project, software developers often work on features that they have an intrinsic interest in or want added to the program. This can sometimes make it challenging to create a cohesive product since it can be difficult to come to a consensus. Frequently, a kind of informal community structure arises. The people with the most experience and those who have substantially contributed to the project have the most influence or power within the community. Since members are often geographically dispersed around the world, communication through the Internet becomes critical for the group to function. This communication network increases in complexity as the number of individuals in the community increases ([Crowston & Howison, 2005](#); [Hinds & Mcgrath, 2006](#)).

2.1.3 Motivation

A significant amount of identity and drive can guide participation in these types of projects (Hertel, Niedner, & Herrmann, 2003; Shah, 2006). Intrinsic motivations consistent with identity in the project, creativity, and social involvement have been found to be more important than extrinsic motivations such as skill development or obtaining a better job (Lakhani & Wolf, 2003). Examples of intrinsic motivation are being a part of the community and the feeling of accomplishment that comes with knowing the software is used by people all over the world. Intrinsic motivation is more related to the intangible social and emotional factors that come into play. Extrinsic motivation is more related to tangible benefits. Examples of extrinsic motivation involve being able to include specific accomplishments on a resume, monetary compensation, or increased technical experience. There are many types of contributors to open source projects with developers having differing levels of personal involvement in the community based on these intrinsic and extrinsic motivations. The extrinsic and intrinsic motivations of open source developers have been shown to be interrelated (Roberts, Hann, & Slaughter, 2006).

Eric Raymond, an open source advocate and programmer believes that the structure of OSS is essentially distributed in nature. This is in stark contrast to proprietary software that is often developed in a hierarchical fashion where programmers work on sections as supervised by their managers. His seminal paper entitled, *The Cathedral and the Bazaar* examined his personal experience with open source software as well as his view of how the structure differs from traditional development (Raymond, 1999). Traditional hierarchical software development was deemed analogous to a cathedral. In this system, programmers were told what to work on by those above them. This monolithic system is contrasted by the open and distributed bazaar software development model. Similar to a crowded market where numerous people are selling a wide variety of items, OSS is similar in that many projects are spun off from one another. Over time, some become more popular while others diminish in popularity or die off.

The question then arises as to why people would want to contribute their time, energy, and skill to a project in the first place. Raymond believes the primary reason that software

developers like coding and working on software is because they like tinkering and the very act of creating something. As he explains, “every good work of software starts by scratching a developer’s personal itch” (Raymond, 1999). It is important to recognize that most of these individuals are not being paid, therefore whatever enjoyment or benefit they are receiving is mostly intrinsic in nature. For these developers, knowing that something they created is being used by people all over the world is a gratifying feeling. To many, showcasing their technical prowess at coding takes a backseat to the socialization and community aspect involved with participating in an open source project. This happens in the same way that someone might volunteer at a homeless shelter and become attached to the community as well as the people being helped. Open source contributors feel a great sense of accomplishment that their product is being used around the world. The core philosophy of open source helps bind these individuals together in a way that they all share a common relationship and ethos.

Another reason people contribute to open source projects is that the experience is an excellent addition to a resume. The social and technical skills developed while contributing to an open source project may help an individual gain recognition for a job or promotion. Many companies that utilize OSS products try to employ key developers and contributors of that project in order to help influence or sway the general direction of the project. Since development is such a collaborative effort, these individuals can push the project towards the plans of the company or at the very least work on features or additions that the company requires.

The success of OSS rests primarily in the fact that the underlying source code is open and can be viewed and modified by anyone. This open development model is better than a closed system because as Raymond explains, “given enough eyeballs, all bugs are shallow”. The more people you have examining and scrutinizing the code, the faster issues will be identified and fixed. In this way, it will be a more stable end product. Certainly, OSS would not have gained in popularity without the Internet. The Internet has shaped not only the communication of members in the community, but also provided a way to easily share code and information around the entire world. Physical locality becomes irrelevant.

2.1.3.1 Psychological theories related to motivation

One motivation to participant in OSS as proposed by researchers is learning (Ye & Kishida, 2003). They have based their work on the theory of Legitimate Peripheral Participation (LPP) (Lave & Wenger, 1991). LPP revolves around the idea that learning is primarily a social activity. Specifically, LPP describes how new developers become experienced members in OSS communities. New developers become members of a community initially by participating in productive yet simple tasks that further the overall project. Through engaging in these peripheral activities, new developers become acquainted with the task dependencies, language and social/power structures of the community. In an OSS domain, LPP assumes that if new developers can observe the practices of established core developers, they will learn how to become productive community members. This implies that, conversely, new developers who are separated from the knowledge and experience of the core developers will have limited access to the tools and community, and therefore will have limited growth (Tuomi, 2000).

The interactants of an open source community and the software that is created is analogous to an onion (see Figure 2.1) with various layers being peeled off to reveal core developers (Crowston & Howison, 2005; Herraiz, Robles, Amor, Romera, & Gonzalez-barahona, 2006; Mockus, Fielding, & Herbsleb, 2002; Moon & Sproull, 2008). The users of a project make up the bulk of the community. The next level within the community are the bug reporters and coders who identify and fix bugs. This is a fairly low level of contribution, however, it is extremely important for the health of the project as a whole. The subsequent levels are developers who spend varying amounts of time on the project. Finally, there is the project leader or leaders who tries to tie everything within the scope of the project together. The interesting part in all of this is that whenever a user makes some sort of contribution, their status and role in the project changes. For example, as a bug fixer fixes more and more bugs, they gain the attention of other developers as well as understand more of the codebase. In this way, they can become a peripheral developer through their contributions.

As Ye and Kishida (2003) explain, “learning in [an open source] community should be viewed as an integral constituent of participation in the community of practice, as a process

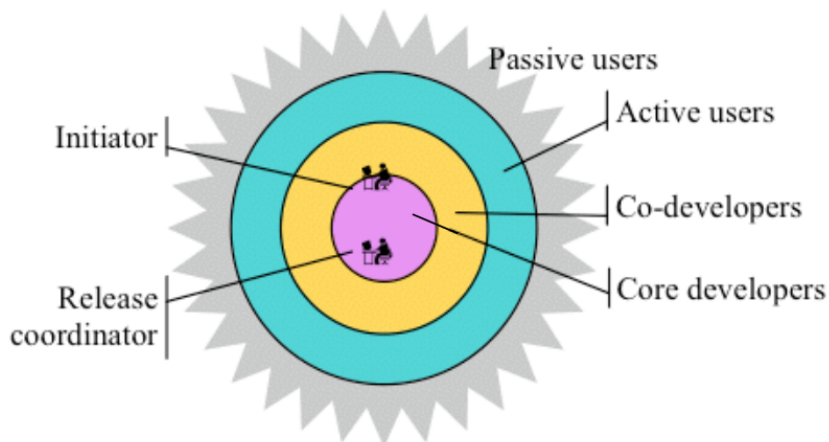


Figure 2.1 Onion Model of OSS (Figure 1 of [Crowston and Howison \(2005\)](#))

of constructing knowing through social interaction with other members of the community, of changing relationships with other members of the community, and of transforming roles and establishing identities from a journeyman to a master in the community”. Communities of practice (COP), “are groups of people who share a concern, set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis” (pg. 4) ([Wenger, McDermott, & Snyder, 2002](#)). It is through this quest for information and creativity that the motivation resides. Open source project participants enjoy tinkering and learning new things. This constant sharing of knowledge fuels the development of the project through exploration. The highly social nature of the project and the constantly changing dynamic of user roles in the community provide a big impact on the group culture and power structure.

As part of this research, a variety of personality factors were examined as possible factors that influence motivation and contributor engagement. Personological variables such as agreeableness and neuroticism may be important factors in how a person interacts with others in an OSS community and the perceived quality of the community regarding entrance and retention of new developers. Agreeableness refers to the extent to which an individual is pleasant and accommodating in social situations ([John & Srivastava, 1999](#)). Individuals high in agreeableness are empathetic, considerate, friendly, generous, and helpful. Neuroticism is the tendency for individuals to experience negative emotional states in a social context ([John & Srivastava,](#)

1999). Individuals high in neuroticism are more likely to interpret ordinary situations as threatening, and minor frustrations as hopelessly difficult, thus making their interactions with others negative.

Rigby and Hassan (2007) wrote one of the few papers examining personality traits of developers in an open source community (the Apache server mailing list). The researchers ran the Linguistic Inquiry and Word Count Tool (Pennebaker, Chung, Ireland, Gonzales, & Booth, 2007) which predicts and classifies words using a known corpus of data. Using sentiment analysis, each word in the corpus is assigned a numerical value corresponding to a positive or negative valence. By creating a baseline on the big five personality traits on the entire mailing list, they found that the top two developers in terms of code commits differed from the baseline on the traits of extroversion and openness.

The prevalence of narcissism and its effects on the community is another factor that could be examined by using the Hypersensitive Narcissism Scale (Hendin & Cheek, 1997). This measure assesses possibly counter-productive individuals in the community. Developers with relatively high scores on this scale could be identified as difficult to work with and more harmful or negative to the community at both the technical and social level.

All human beings have a feeling that they need to “belong” to specific social groups based on their own core identity. In order to assess the extent to which developers use OSS community participation to fulfill this need, assessment of individuals using the Need to Belong Scale (Leary, Kelly, Cottrell, & Schreindorfer, 2007) could be utilized. This measure contains items that assess the general extent to which people feel the need to be included in groups.

Need for cognition is the extent to which individuals are motivated to thoughtfully consider information and it has been shown to be an important factor in social influence processes (Cacioppo, Petty, Feinstein, & Jarvis, 1996). Specifically, Shestowsky, Wegener, and Fabrigar (1998) found that individuals high in need for cognition were better able to influence their dyadic partner than those low in need for cognition. For new developers in an OSS community, need for cognition may help predict how well the existing community accepts their ideas, in part because individuals who are high in need for cognition are better able to influence others.

McCroskey (1992) developed the Willingness to Communicate (WTC) scale that measures how willing or unwilling someone is at communicating. Subscales measure more specific situations such as group meetings or public speaking, as well as smaller settings such as talking with a friend or acquaintance. The survey assumes that participants are aware of how willing or not they are to communicate and contrasts with personality methods that attempt to assess introspective or extroverted individuals. A potential downside of this survey is that all the questions are related to in-person communication with no questions tailored to Internet communication.

Variability in subjective amounts of perceived investment and commitment in the OSS project may help predict how successful the project is, and how efficient the community is. Previous research has found that people who are highly committed and invested in an organization are less likely to quit an organization (Pfeffer, 1983). New developers who are committed and invested in the community should be more likely to be a member through the project's completion. Commitment and investment in both the project and community could be assessed.

Communication is in part determined by the memory interactants need for relevant information (Holtgraves, 2010). The more similar the information one has about a topic or object, concepts related to the topic are more easily transferred across interactants (Horton & Gerrig, 2005). In other words, communication plays a decisive role in the development of shared or transactive memory systems (Palazzolo, Serb, She, Su, & Contractor, 2006). Transactive memory is a specific type of memory where one's memory is influenced by knowledge about the memory system of another person (Wegner, Erber, & Raymond, 1991). Members working on interdependent tasks may communicate effectively in part because they have similar memory systems (Espinosa, Slaughter, Kraut, & Herbsleb, 2007b). While much of the work on the utility of transactive memory has been conducted on small face-to-face groups (Ilgen, Hollenbeck, Johnson, & Jundt, 2005), a number of studies have demonstrated a positive influence of transactive memory systems on large and geographically distributed groups (Espinosa, Slaughter, Kraut, & Herbsleb, 2007a).

2.1.4 Tools

Open source projects currently utilize a wide variety of technical tools to propel their projects forward. Many tools such as source code version control focus on working on source code in a decentralized manner. Other online tools take suggestions and comments from users. The most important tool is the primarily text based communication that takes place over the Internet.

2.1.4.1 Source Code Version Control

Source code version control software helps keep track and manage source code across a group of developers. When developers make changes to files, these changes are then made available to other developers in the community. In this way, multiple developers can be working on different parts of the project all at the same time. Source code version control also keeps track of the history of all changes made which allows developers to experiment with new code and return to working code if their new code does not work as intended. Other tools allow merging source code together as well as dealing with conflicts if two developers are working on the same file at the same time.

Examples of source code control software are Subversion (SVN), Concurrent Version System (CVS), Bazaar, Mercurial, and Git. There are two main types of version control software, centralized and decentralized. Centralized version control such as SVN or CVS has a central server that all code is “checked-in” to and saved. Any commits that are saved have to be checked against this central server. Decentralized, also called distributed version control, allows commits to be made locally and then pushed to one or more remote servers. This distributed nature of the software makes it easy for individuals to try and test code before it goes through a group review process. Bazaar, Mercurial, and Git are all examples of distributed version control systems. Git in particular was created by Linus Torvalds to be used specifically for Linux kernel development but has since expanded to become one of the more widely used version control systems by software developers.

The website Github ([Github, 2015](#)) has become incredibly large and influential in hosting source code repositories of open source projects. Traditionally, in software development, forking has been considered poor practice because you are splitting away and not pooling effort. Github turns this idea around and makes it incredibly easy to view other people's code and fork it into your own repository. You can then make the changes or bug fixes that you identify. The benefit is that everyone can see this fork and other developers can then merge the changes that you have made back into their projects. Along with a code review process for managing pull requests, commenting, and other social communication features, Github provides substantial value.

2.1.4.2 Bug Tracking

Bug tracking software handles reporting and management of bugs within software. A bug is an error or mistake in the source code that is often difficult to detect without actually running the program. There can be syntactic bugs such as a programmer missing a semicolon. This type of bug results in a compiler error or run-time error if the programming language is interpreted. Usually these types of bugs are easier to fix because the compiler or interpreter provides a line number indicating roughly where the fault exists. Run-time errors are more difficult to diagnose. For example, if a user inputs something unexpected by the programmer, this could result in an error or issue later on. Run-time errors are not caught by compilers and thus are more difficult to debug and diagnose. Developers and users of the software find problems and then report them using a bug tracker. This then helps the developers understand the issue and the severity of the problem. With this information, they can attempt to determine where the bug is within the code and resolve the issue. After repairing the bug, the issue is considered resolved and archived so that other developers and users can see that the issue was fixed.

Some examples of bug tracking systems are Bugzilla, Mantis, Trac, Redmine, and Jira. These require a webserver and a database for setup and use. Various online systems provide pre-configured systems for bug tracking and overall project management such as Canonical's Launchpad, Github, Sourceforge, and Assembla.

2.1.4.3 Continuous Integration

Continuous integration is the process of building and testing the source code of a project to identify compile errors and run unit tests. For projects that provide rapid deployment of new versions of their software to users, this can be incredibly valuable. A server is setup that runs continuous integration software that can either be triggered manually by developers or automatically based on version control commits. When the server is triggered, a build is started and the program is compiled and unit tests are run. If there is a compile error, this then alerts the developers that there is a problem with the source code. Unit tests are source code that test the functionality and assumptions of the source code. As part of test-driven development, they provide a way for developers to ensure that future changes won't break the overall system. Starting with writing unit tests helps developers understand the product requirements and think about how the piece of code fits into the larger system. Subsequently, developers will write the source code, and then run the unit tests to ensure the test passes. Through iteration of the unit tests and source code, they can add new functionality to the overall system. Successful unit tests can help alleviate potential run-time errors before they get to the end user.

2.1.4.4 Communication

Without the Internet, it is clear that OSS would be vastly different, if it even existed at all. The Internet has provided the ability to communicate cheaply, quickly, and effectively with large numbers of users in geographically diverse locations. Most open source projects utilize text based communication messages. Email mailing lists and forum message boards are two of the most popular Internet communication mediums. Through these mediums, discussions take place on the direction and scope of the project. Internet Relay Chat (IRC) and instant messaging also provide immediate real-time chat and availability. Instead of reading through a mailing list and trying to find the answer to a question, users can directly communicate with developers of the project in a group setting online. Most of the communication methods chosen by OSS developers are text based. This leads to both problems and benefits. The richness of

text based communication as a message passing medium is not as strong as other communication methods. As McLuhan proposed in a famous phrase, “the medium is the message” (McLuhan, 1964). This infers that the medium itself, and not the content of the message is important and should be studied. Upon inspection, text based communication has an interesting advantage. Since users in a mailing list are composing a message that many people will see, they tend to think very carefully about the message and the content before sending it. They are often very focused on the task at hand and the technical details are precise. Messages are concise and to the point. The developer sending the email wants to make sure other developers are reading this email. By taking time and thinking carefully about the message before hitting the send button, the developer is able to use this text-based communication medium effectively.

There is also a possibility that the type of feedback (positive vs. negative) might be interpreted or used differently by beginners versus veteran members. In a study comparing novices and experts in a variety of fields (none were software development), researchers found novices responded more to positive feedback whereas experts responded more to negative feedback.

“We predict an increase in negative feedback as people gain expertise, because the meaning people derive from feedback changes such that negative feedback increases the motivation to adhere to a goal. In support of our prediction, we find that novices infer from feedback whether their goals are valuable (commitment), whereas experts infer from feedback whether their pace of pursuing already valuable goals is sufficient (progress) (pg. 15) (Finkelstein & Fishbach, 2012).”

The researchers appear to be arguing that negative feedback can be beneficial or at least stimulate more feedback in certain situations.

2.2 Social Technical Congruence (STC)

Understanding the organization and structure of open source projects is helpful in understanding group dynamics at work, as well as how these systems can produce effective end results. As stated earlier, the traditional model of software development is hierarchical. In the traditional model, the work is mostly distributed to low level coders. These coders receive

instructions and support from developers higher up in the system. This progresses upward to the top similar to a tree structure where the head of the project makes all high level decisions. This traditional software development practice is still predominantly used today. In contrast, distributed development is where there is no set hierarchy of reporting. All users interact and share information with one another. In this way, all users can contribute to different sections and provide insight to various parts of the project.

2.2.1 Classical Hierarchy

While many people have come up with explanations to hierarchical systems, Max Weber was probably the first to examine a traditional hierarchy. A German sociologist, Weber created the Theory of Bureaucracy to help managers run the organizations under them. His writing on the subject can be broken down into the following six categories (pg. 11- 12) ([Miller, 1998](#)).

- An organization needs to have a clearly defined hierarchy.
- The organization has a clearly defined division of labor.
- There is a centralization of decision making and power within the system.
- These organizations are closed systems that are not influenced by outside forces or the environment.
- Rules need to be established to help keep the organization functioning and in correct order.
- The system is based on the functioning of an authority of power and discipline of which there are three possible types:
 - Traditional authority is power based on long standing beliefs about who should have control.
 - Charismatic authority is power based on an individual's personality and ability to attract and interact with followers.

- Rational-legal authority is power based on the rational application of rules developed through a reliance on information and expertise.

Most companies that develop proprietary software products follow the majority of Weber's principles. They follow the hierarchical model as it relates to power and software development organization.

2.2.2 Distributed Structure

Open source software is often described to be distributed in nature with little or no hierarchy of structure. This viewpoint was taken by Eric Raymond in his *Cathedral and the Bazaar* paper (Raymond, 1999). However, after examining open source projects, the relationships and power structure between users is dynamic and constantly changing (McGrew, Bilotta, & Deeney, 1999). OSS is clearly different than traditional software development with the primary difference being developers can work on whatever section interests them. This should be considered the “distributed” aspect of open source software.

Some researchers have tried to combine a variety of aspects of OSS development into one framework. Sack et al. (2006) looked at the, “social, technical, and cognitive aspects of OSS development”. By using a variety of methodologies, they found that their work supported a theory of the relationship between the social structure of the group and the underlying structure of the development process. Dubbed, “Conway’s Law” (Conway, 1968), “the governance structure of the project (e.g., who manages whom, who communicates with whom, etc.) has a direct influence on the structure of the software itself (e.g., its division into modules). [It] was the first explicit recognition that the communication patterns left an indelible mark upon the product built” (Sack et al., 2006). The authors contend that open source projects might have a reverse Conway’s Law where, “the technical structure of the software might directly influence the social and governance structure of the project” (Sack et al., 2006). While this is something that requires further analysis, it appears that the communication and modularity of the project are correlated. Strohmaier and Wermelinger (2009) found that software artifacts

that were highly connected to other software artifacts required more developers to look at bug reports and feature requests thus supporting the concept of Conway's Law.

Modularizing a project into sections is a critical component of collaborative efforts and can result in shorter development time (Parnas, 1972). Open source projects exhibit the, "small world" phenomenon (Amrit, Hillegersberg, & Kumar, 2004) pioneered by Milgram (1967). Popularized through the phrase, "six degrees of separation", the phenomenon contends that human society is highly connected with individuals needing only a few connections to reach a known person. Open source development can be seen as analogous to a living and evolving entity. The constant development means small changes over time shape the overarching structure of the entire software system as well as the community.

Open source projects also seem to follow the pareto principle, also known as the 80-20 rule. This means that 20% of the developers are doing roughly 80% of the work coding (Robles & Gonzalez-Barahona, 2006; Robles, Koch, & González-Barahona, 2004). The 80-20 rule relates to a power law or long tail distribution where the distribution has high numbers of occurrences in the tail of the distribution.

"F/OSS [free and open source software] systems seem to evolve through minor improvements or mutations that are expressed, recombined, and redistributed across many releases with short duration life cycles. Foss end users who act as developers or maintainers continually produce these mutations... As a result, these mutations articulate and adapt an F/OSS system to what its user-developers want it to do while reinventing the system... F/OSS system co-evolve with their development communities; one's evolution depends on the other's" (Scacchi, 2004).

The actual development is not a strict hierarchy or a completely decentralized system. It is a hybrid that seems to merge the two systems together. Scacchi (2004) dubbed it an, "interlinked layered meritocracy".

"FOSS development teams can take the organizational form of interlinked layered meritocracies operating as dynamically organized but loosely coupled virtual enterprise. A layered meritocracy is a hierarchical organizational form that centralizes

and concentrates certain kinds of authority, trust, and respect for experience and accomplishment within the team. [M]eritocracies tend to embrace incremental innovations, such as evolutionary mutations to an existing software code base, over radical ones” (Scacchi, 2004).

The lowest level in the pyramid and the most numerous are the regular users. The next level are the bug fixers and main developers. Finally, at the top layer are the core developers. Often, these core developers are the individuals that started the project. In this way, as a contributor increases time and commitment to the project, their status also increases and they have more say in the community.

2.2.2.1 Social Technical Congruence (STC)

In software development, the structure and dependencies of source code as well as the relationships and communications between developers has been shown to be inextricably tied. The term, Social Technical Congruence (STC), emphasizes the relationship between the code structure and communication structure. STC represents the overlap between social communication and an organization’s coordination of tasks (Cataldo et al., 2006). Even before STC was coined, researchers were understanding the importance of dependencies in complex systems (Krackhardt & Carley, 1998). Understanding these relationships aids in understanding a group’s productivity (i.e., task efficiency, overall time developing a project, etc.).

Consistent with the STC perspective, when the congruence or “fit” between the task dependencies and the coordination activities of group members is high, then the project is healthier and has a shorter development time (Cataldo, Herbsleb, & Carley, 2008; Cataldo et al., 2006). In test driven development, high congruence has been positively correlated with continuous build success rate (Kwan, Schröter, & Damian, 2011). However, for integration builds in which code from multiple sites is merged together and tested, congruence was found to be negatively correlated with build success.

Although intuitive, there are a number of variables that have to be considered when calculating congruence (Sarma & Herbsleb, 2008a). There can be costs associated with changing

source code or communication structure to increase congruence (Valetto, Jose, & Williams, 2008). Sifting through large amounts of email to understand overall code changes as well as the design goals going forward takes time and diverts attention away from the overall project. Thus, developers need to be able to communicate effectively with the right people, while at the same time discouraging communication overload.

In addition, the modularity of the code structure is one of the driving forces behind the need for STC in software development. As developers work to modularize the project as a whole, one consequence is that dependencies between modules are created. This understanding of the overall dependency structure of the project is critically important. Some pieces of code may be crucial for the entire project while others may be less important. As developers make changes to the code base, they need to understand how these changes affect dependent modules, as well as effectively communicate these changes to the correct developers. If communication is missing and developers are not interacting with one another but are both working on modules that are dependent on one another, serious problems can arise (see Figure 2.2). The same scenario can be applied to complex system design.

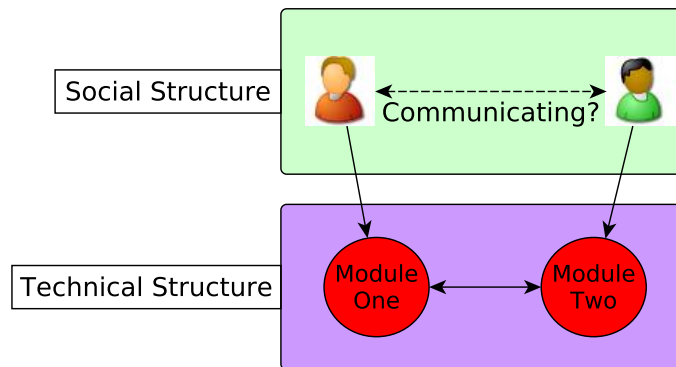


Figure 2.2 Communication Between Developers of Dependent Modules

Open source projects provide excellent test-beds for understanding the social and technical aspects of complex system design because of the large amount of historic and real time information that is available. Text-based communication among developers and elaborate source code changes available through version control systems make OSS communities a popular choice for researchers studying social and structural elements of communication (Sack et al., 2006), es-

pecially in improving existing STC algorithms (Herbsleb, Roberts, & Mockus, 2006; Herbsleb, Sarma, Mockus, & Cataldo, 2008; Kwan, 2011; Kwan, Schr, & Damian, 2009; Sarma & Herbsleb, 2008c). This research leverages this rich accessible data set to develop STC methods and apply those methods to facilitate effective integration of new members into OSS communities.

2.2.3 STC Algorithm Example

The following is a simple example to show how congruence is quantified and calculated. Figure 2.3 shows three developers and four code artifacts. The arrows represent communication between developers, code artifacts that developers are working on, and dependencies between code artifacts. Developers are denoted by alphabet characters and code artifacts are denoted by numerical values.

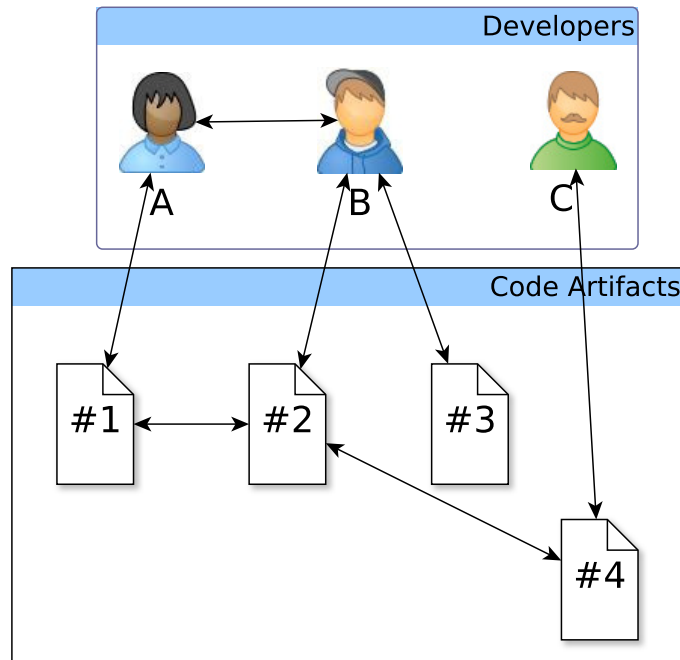


Figure 2.3 Network Structure Between Developers and Source Code

Given n developers and s code artifacts, coordination can be defined as:

$$Coordination = A x D x A^T \quad (2.1)$$

A is an $n \times s$ matrix assigning developers to code artifacts.

D is an $s \times s$ matrix assigning technical dependencies between code artifacts.

Coordination is an $n \times n$ matrix of developers representing who should be coordinating given shared artifact dependencies.

By taking the difference between this coordination result and the actual communication that is occurring, the result is the missing communication between developers. Congruence is represented as the ratio of actual communication over the amount that should be occurring and has a range from 0 to 1. Calculation of congruence using this simple example is provided below.

First, a developer by code artifact matrix is created, this represents the artifacts developers are working on.

$$\begin{array}{c} \\ \\ \\ \end{array} \begin{array}{cccc} 1 & 2 & 3 & 4 \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \end{array}$$

Next, a code artifact by code artifact matrix is created, this represents the dependencies between these sections of code.

$$\begin{array}{cccc} & 1 & 2 & 3 & 4 \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \end{array}$$

By multiplying the two matrices, a developer by code artifact matrix is created which represents code that developers should be aware of.

$$\begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 1 \ 2 \ 3 \ 4 \\
 \left[\begin{array}{cccc}
 1 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 0 & 0 & 0 & 1
 \end{array} \right]
 \end{array}
 x
 \begin{array}{c}
 1 \\
 2 \\
 3 \\
 4
 \end{array}
 \begin{array}{c}
 1 \ 2 \ 3 \ 4 \\
 \left[\begin{array}{cccc}
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 \\
 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 1 \ 2 \ 3 \ 4 \\
 \left[\begin{array}{cccc}
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 \\
 0 & 1 & 0 & 1
 \end{array} \right]
 \end{array}$$

This result is then multiplied by the developer by code artifact (task assignment) transpose. This gives the coordination requirements matrix which shows developers who should be communicating based on shared dependencies.

$$\begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 1 \ 2 \ 3 \ 4 \\
 \left[\begin{array}{cccc}
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 \\
 0 & 1 & 0 & 1
 \end{array} \right]
 \end{array}
 x
 \begin{array}{c}
 1 \\
 2 \\
 3 \\
 4
 \end{array}
 \begin{array}{c}
 A \ B \ C \\
 \left[\begin{array}{ccc}
 1 & 0 & 0 \\
 0 & 1 & 0 \\
 0 & 1 & 0 \\
 0 & 0 & 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 A \ B \ C \\
 \left[\begin{array}{ccc}
 1 & 1 & 0 \\
 1 & 1 & 1 \\
 0 & 1 & 1
 \end{array} \right]
 \end{array}$$

At this point, the difference between this result and the actual communication that is occurring is the missing communication between developers.

$$\begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 A \ B \ C \\
 \left[\begin{array}{ccc}
 1 & 1 & 0 \\
 1 & 1 & 1 \\
 0 & 1 & 1
 \end{array} \right]
 \end{array}
 -
 \begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 A \ B \ C \\
 \left[\begin{array}{ccc}
 1 & 1 & 0 \\
 1 & 1 & 0 \\
 0 & 0 & 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 A \\
 B \\
 C
 \end{array}
 \begin{array}{c}
 A \ B \ C \\
 \left[\begin{array}{ccc}
 0 & 0 & 0 \\
 0 & 0 & 1 \\
 0 & 1 & 0
 \end{array} \right]
 \end{array}$$

Based on this result, developers *B* and *C* should be communicating because of shared dependencies in code artifacts #2 and #4, but they are not. A single global value of congruence can be calculated as a ratio. In the above example, two pairs of developers (*A* and *B*, *B* and *C*) should be communicating and of those only one pair (*A* and *B*) are actually communicating. The ratio would be $1/2 = 0.5$. It is the proportion of communication that occurred relative to the amount that should take place. High congruence values show that the necessary communication is taking place and research has shown that they yield a shorter development

time (Cataldo et al., 2006). Low congruence values suggest people are not communicating enough.

In the above example, the edges or dependencies in the graph are denoted as binary, the edge is either present or absent. Weighted edges have also been successfully applied to the calculation of congruence which can lead to a more granular and detailed representation of the data (Kwan et al., 2009).

The idea of leadership has also been applied to the calculation of congruence (Dubinsky & Hazzan, 2008). In this work, the idea of a *Methodology Change Leader* (MCL) is posited. This person serves as a knowledge center and is aware of the availability of information and current problems. They are also the leadership center in that they delegate team members to different tasks. And finally, they are a communication relay, especially between different groups of people (developers and software users for example). Based on the communication and knowledge that a leader has, they are aptly fit to be considered in the calculation of congruence. While the authors do not go into detail into how this would be accomplished, it is certainly something that can be applied as part of the model.

Development experience is another factor that has been applied to the idea of STC (Sarma & Herbsleb, 2008c). In this work, researchers use the concept of a shared mental model to identify coordination level. A shared mental model matrix is created where elements represent the amount developers have worked together in the past. In addition, expertise congruence is identified as the match between the expertise and knowledge necessary for the project versus the amount of expertise that is actually available.

Congruence has also been framed as a distributed constraint satisfaction problem (DCSP) (Herbsleb et al., 2006, 2008). The construction of a piece of software is completed by making engineering decisions. Constraints limit the decisions that can be made. “Successfully completing a project is equivalent to finding an assignment for all variables that satisfies all constraints” (Herbsleb et al., 2008). While theoretically feasible, this framework may be more difficult to apply practically to communities in the real world.

Valetto et al. (2007) explain a similar system of calculating congruence as Cataldo et al. (2006), however their method uses digraphs and sets as definitions. Another method of congru-

ence calculation is able to calculate an individualized congruence value (Wagstrom & Herbsleb, 2008). In this way, each developer can have a specific congruence value that is relative to them. They found that higher individual congruence lead to higher performance in terms of bug fixes and the time required to resolve the bug. Developers who had higher levels of required communication took longer in resolving bugs.

One of the open questions of social technical congruence is what communication gaps represent. Does the identification of a communication gap mean that people are using different informal channels of communication to transmit relevant knowledge? Perhaps the communication medium being used does not exist in the model. Do they simply not feel the need to communicate because they have all the necessary information? What about extra communication? Is communication between individuals who are not working on dependent code artifacts unnecessary? One research group analyzed congruence gaps and found that developers who were identified as having gaps in communication were associated with increased amounts of code changes as measured by the commit history (Ehrlich, Helander, Valetto, Williams, & Davies, 2008). They also found that information brokers played a key role. Brokers are people who are highly connected in the graph network and are often connected with other highly connected individuals. In this way, they act as information hubs and are able to pass on relevant information to other groups. In online communities, this can be incredibly important because these are the people that have the social and political power in the community. They might even spend more time managing people and the direction of the community than they do coding on the software. Valetto et al. (2008) discuss various options available regarding how to close congruence gaps between developers as well as the side effects that crop up as a result of these actions. Obviously closing communication gaps is meant to ameliorate a problem, however, forcing communication costs time and energy. As they explain, “eliminating a congruence gap by closing it may produce a net benefit in terms of project value only when the value gained due to the increased congruence is likely to offset all the additional costs sustained because of that communication channel” (Valetto et al., 2008).

A critical aspect of congruence and modeling these dynamic online communities is understanding how the passage of time changes the community. For source code, it can “decay” in the

sense that developers feel it is more difficult to maintain and manage as time progresses (Ohlsson, Mayrhauser, McGuire, & Wohlin, 1999). Eick, Graves, Karr, Marron, and Mockus (2001) outline various properties to take into account when measuring code decay such as the number of changes and what the change constitutes to the file, the date, and the size of the codebase. Through an analysis of a large codebase they found that code does indeed seem to decay over time.

Another concept of time and decay has been applied to communication and coordination. Wagstrom, Herbsleb, and Carley (2009) tweaked the model of STC such that links and dependencies in the graph are adjusted through a decay factor over time. Thus older communication and dependencies are removed from the graph as time passes. Validation was done by running the algorithm against projects in the Gnome community.

2.3 Socialization of New Developers

Despite the utility of creating products and services from an open source format, this type of model is not without its weaknesses. One drawback in particular that can affect a project's effectiveness is the existence of a significant barrier to entry for new developers wishing to join the community (von Krogh, Spaeth, & Lakhani, 2003). New developers can provide ideas, skills, and suggestions, and could eventually become core developers. However, new developers need to also have an understanding of the social community in order to communicate their skills and ideas effectively (Ducheneaut, 2003).

Examining newcomer socialization (Burke, Kraut, & Joyce, 2010; Kraut, Burke, & Riedl, 2011) and group dynamics in online communities other than OSS have been fruitful. For example, topics such as leadership behavior on Wikipedia (Collier, Burke, Kittur, & Kraut, 2010); social capital and well-being (Burke, Marlow, & Lento, 2010), and motivation (Burke, Marlow, & Lento, 2009, 2010) on social network sites such as Facebook; and even politeness in online community forums (Burke & Kraut, 2008) have emphasized the importance of communication in group harmony and the health and well-being of the community. In a traditional for-profit software development company, new developers were examined and it was found that the three factors that impacted their successful integration into the community were: early experientia-

tion, internalizing structures and cultures, and progress validation (Dagenais, Ossher, Bellamy, Robillard, & de Vries, 2010). The proposed research builds upon this previous work by examining socialization in the development and evolution of an OSS community and its influence on new developer retention and community effectiveness.

New developers are different than existing members of the community in a number of ways. For one, new developers do not possess the knowledge regarding the historical development of the social structure of the community prior to their entrance into the group. To help increase socialization in joining the group (Berger & Calabrese, 1975), new developers will watch or “lurk” on the email lists to monitor what the community is discussing as well as understand who has power and influence in the community. The downside of this, rather indirect, assessment of community dynamics is that it can take a long time to learn the community structure (Lave & Wenger, 1991), and only those who are willing to communicate (Sllinen-Kuparinen, McCroskey, & Richmond, 1991) may reach out to this new environment. Given that social exchanges among new and existing members can lead to an increase in a new developer’s performance (Chen & Klimoski, 2003) and the likely impact these have on the health of the project, it seems prudent to help understand the social-communicative aspect of new and existing members.

Most communication in an OSS community is conducted through email, mailing lists, forums, live chat, or other text-based communication services. Text-based communication is used more than voice-chat or live video streams because developers are not always online at the same time. Asynchronous methods of collaboration are used. In addition, text-based communication yields a record of the conversation which is easily search-able. One of the downsides of this informal communication process for new developers is the difficulty in determining who to contact as well as a potential lack of trust or openness (Herbsleb & Grinter, 1999). Since there is limited face-to-face physical interaction, knowledge of other members is primarily reflected in the quality of their code submissions and their commitment to the project as a whole (Ducheneaut, 2005). Knowledge of the group members’ social customs, emotionality, actions, and goals are important indicators of group cohesion and effectiveness (McGrath, 1997), but are virtually nonexistent in these text-based communication contexts. On the other hand, this reliance on text-based communication levels the playing field for power and dominance in the

group (Dubrovsky, Kiesler, & Sethna, 1991; McGuire, Kiesler, & Siegel, 1987; Sproull & Kiesler, 1986). Who a member communicates with, the tone of the message, their coding skills, what they are working on, and their expertise in certain areas are all considerations that influence the perception of that member to the group (Fogel, 2005). The culture and reliance on email also results in the community being highly rational in the design making process (Dubrovsky et al., 1991; Yamauchi, Yokozawa, Shinohara, & Ishida, 2000). Since there are diminished social cues, the only criteria for making a decision is in whether or not it is faster/smaller/easier from a technical perspective.

Email lists provide a vast amount of quantitative information that some researchers have examined. Researchers devised a study where they examined two large open source project email lists and categorized the emails into five different groupings (Yamauchi et al., 2000).

- Asking for information
- Responding to messages / Giving information
- Proposing an idea
- Reporting results of action
- Other

They found that giving information accounted for about half of all email messages. Reporting results and asking for information were next with about 20% and 15% of the messages falling into these respective categories. It was interesting to note that only around 7% of the messages dealt with proposing ideas. These mailing lists seem to rely more on contributors coming forward and showing their progress as opposed to talking about something that could be built or added. In this way, they are “biased towards action rather than coordination”. “The report of action triggers discussion more frequently than discussion initiates action” (Yamauchi et al., 2000). The researchers found that about 75% of tasks were started without being declared and about 58% were finished without the contributor even declaring he or she was working on it. This is interesting because it seems to be contrary to the social and collaborative

nature of OSS. The act of coding appears to be a very private a solitary affair but once the code is finished and shown to the community at large, the social aspect of the community arises by examining and scrutinizing the code.

The source code itself can also be seen as a message. Source code often contain comments which are messages to source code readers that are not actually part of the program itself. These comment notes are written by the programmer to explain certain sections of code. An examination of who is contributing code and where the contribution is can communicate the interest level of each developer. In addition, since the source code is open and can be examined by other developers, the technical prowess of a developer can be ascertained.

Some larger projects have yearly meetings to bring key project contributors together to discuss the project, however, this is often limited to projects with corporate backing. The Ubuntu Linux distribution for example has an annual meeting where key developers and contributors select a location to meet and discuss the future of Ubuntu. Text based communication is effective, however, being physically present provides an added benefit both in terms of getting the message across, fostering collaboration, and providing non-verbal physical communication.

New developers often lack the communication skills to effectively pitch their ideas. If a new developer posts a message requesting to completely change the system with no regard for the current structure, this new idea will likely be rejected. Members who are able to communicate effectively, to rally support for issues they feel are important, and to show their expertise, are able to influence the perception of the group. Successful changes or modifications in the community require both fitting in with the existing project scope and timeline, as well as allies within the community who are powerful enough to support the change. Thus, the successful socialization of a new developer is a process ([Hollander, 1958](#)). OSS development can be a political process and new developers need to understand how to effectively navigate this in order to be successful.

New developers also present challenges to the community because they take away time from other members who could be working on code. [Collins-Sussman and Fitzpatrick \(2007\)](#) explain various ways in which individuals wishing to join a community can go about this in the wrong way and become, “poisonous people” that may slow down or derail a project. New developers

need to be humble and courteous and realize that they are joining an existing community with a power structure they know little about (Fogel, 2005). Being brash or asking too many questions early on will alienate them from the community and be a black mark on any future participation they may have. These actions can either be intentional or unintentional but the end result is still negative for the health of the project as a whole. The most important resource the project has is the attention and focus of the developers. New users are often considered poisonous because the time investment of developers on new users does not guarantee that they will become core contributing members of the community. There is a certain level of trust required when a new member says they are going to implement something or work on a feature. The community has to trust that they are actually going to proceed with the work.

Running a successful open source project can be a challenging task. The complexity of the system means that small changes can have big impacts. Karl Fogel created a document to help people understand how open source works, as well as his suggestions for managing a successful open source project. When examining the health of a project, one has to make sure that the project is able to accept new users, contributions, and bug reports. In addition, it needs to be able to survive if key members of the project leave.

As Fogel explains, “there are various ways to achieve this kind of success. Some involve a formal governance structure, by which debates are resolved, new developers are invited in (and sometimes out), new features planned, and so on. Others involve less formal structure, but more conscious self-restraint, to produce an atmosphere of fairness that people can rely on as a *de facto* form of governance. Both ways lead to the same result: a sense of institutional permanence, supported by habits and procedures that are well understood by everyone who participates. These features are even more important in self-organizing systems than in centrally-controlled ones, because in self-organizing systems, everyone is conscious that a few bad apples can spoil the whole barrel, at least for awhile” (pg. 56) (Fogel, 2005).

Those in the community need to have faith that the project is stable and willing to take their input into consideration. If not, they may leave the project entirely or fork the project.

Many times if there is a heated disagreement within the community about a planned feature or change to the project, a group of people will disband from the community and create their own separate group. In some situations this is negative, and in others, it is positive. This is just one way that the project can change and evolve. The community climate and culture is extremely important in gauging the health of an open source community.

2.3.1 Climate and Culture

There is a significant difference between the climate of a community versus the culture of a community. The climate of a community is more volatile and often changes day by day. It represents the overall mood of the constituents. The culture of a community is much more stable and long term. It is highly influenced by the past, especially the founding members of the community. The culture can represent the commonality between members both in terms of their viewpoint on a matter, as well as the skills they bring to that community.

The communication in an open source project can change the climate of the project very quickly. A particularly disgruntled user can negatively sour the mood in an email list or a recent development by a coder can positively solve a problem that has been plaguing the project for weeks. Since there are usually no deadlines or consequences for not providing something to the project, the pressure is relatively low. As one developer put it, “I’m always relaxed. I can fail because I don’t tell others what I do” (Yamauchi et al., 2000). One interesting element about the culture of open source projects is that logical decisions seem to take precedence.

“Members try to make their behavior logically plausible and technologically superior options are always chosen in decision-making... The reduction of social context cues by electronic media equalizes the participants: Even a CEO and a high school student can have the same presence. The authority does not work in this situation. In addition, computer-mediated communication is inherently impersonal and prompts task-oriented and focused exchanges” (Yamauchi et al., 2000).

Developers are focused on the task at hand when sending emails and this helps by keeping communication concise. Arriving at a rational decision by examining the technical details of

a new feature or change means that in some cases, decision making is fairly straight forward. Most of the discussion and indecision stems from areas such as usability, interface design, or the future direction of the project.

When communicating with others, it is important to realize that a person may be technically competent but if they do not have sufficient communication skills, their contributions may not be as well received or adopted. Since almost all communication through the Internet is done via text-based messaging, writing properly and effectively is incredibly important. “You may be brilliant, perceptive, and charismatic in person- but if your emails are rambling and unstructured, people will assume that’s the real you” (pg. 77) (Fogel, 2005). Fogel gives a fantastic example where a developer in an FSF project was posting great code and was really able to effectively communicate issues and changes in the code. None of the other developers had met this person. There were legal issues with taking code from strangers so the FSF contacted this developer to sign some legal documents. As it turned out, the developer was a thirteen year old living with his parents. This equalization of participants through electronic communication meant his highly detailed and eloquent posts did not give away his age.

In using text-based communication, it is important to consider the structure of the message. The message needs to flow easily as well as be precise. The tone of the message is also very important. Since messages are often very technical and short in nature, this leaves little room for pleasantries or feelings. Depending on the individual, some community members may think the messages are rather rude. When writing a message, the writer has to take into account past experiences with the recipient of the message to gauge the best way to broach the subject. Being able to effectively read and communicate with others helps a developer understand the climate and culture of the community and overall project.

The culture of an open source project is also tied to the open source movement in a more general sense. These users often have an understanding of the history surrounding OSS and its key philosophies. While some are more vocal and staunch than others with regard to core principles, this commonality helps bind them together. This communication and shared viewpoints influence the overall culture of a project, as well as the day to day climate. The

structure, culture, and communication come together to form relationships between project members and establish power in selected members.

2.3.2 Relationships and Influence

Through this large amount of communication, relationships will inevitably form and a power structure will emerge. The more time and energy a person invests in a project, the higher the influence. Core developers will become highly connected with other core developers through email interactions and code contributions relative to other roles. “These intangible resources (or social capital) arise in many forms. They include (a) assuming ownership or responsibility of a community software module, (b) voting on the approval of individual action or contribution to ongoing project software, (c) shared peer reviewing, and (d) contributing gifts that are reusable and modifiable common goods. They also exist through the project’s recognition of core developer’s status, reputation, and geek fame” (Scacchi, 2007). By building up relationships and thus power with other members, this can help an individual developer spread his or her ideas about the project and future code. This “geek fame” that Scacchi mentions is interesting because it is highly related to the intrinsic rewards that motivate a developer to contribute to an open source project. For many people, recognition and fame can come from participation in OSS. The project receives a high level of support and time from the developer and the developer is positively rewarded for participating. There are benefits for both parties.

The beauty of the entire system is that these relationships and power levels between users are constantly changing. A developer in a project needs to have technical abilities as well as clearly developed communication skills to interact with others. “In online communities, control over artifacts is a central source of power and influence, and textual resources are manipulated to project a participant’s ‘virtual’ identity. Contributing to the creation of an artifact is, therefore, as much a concrete activity of production as a social act of identity building” (Ducheneaut, 2005). An artifact in this case might be a piece of code or an idea for the future direction of the project. Being a successful communicator means not only conveying an idea to the community but gaining the support of other members in that community. By aligning their interests with

that of a developer, a contributor can increase their influence. Becoming a core member of the community does not happen overnight. Given the ever changing relationships, new users who want to help contribute to the project can have a difficult time joining the community.

2.3.3 Socializing New Members

Helping new users become socialized into the project community is an important facet of open source development. There is often constant turnaround as developers come and go, working on the project in an irregular fashion. New members can provide new ideas, skills, suggestions, bug fixes, and features. In a general sense, socialization refers to the process of assimilating someone new through cognitive and behavioral processes into a community. The community as a whole is shaping this new user over time. In addition, this new user is having an impact on the community as a whole. There are a variety of phases to socialization as a new developer enters an open source community.

Before joining the community, an individual has anticipatory socialization in which they have certain assumptions regarding how they will fit into a specific community, as well as what efforts are required to gain this status and their role. This is also the stage where they learn more about the community by reading about it and often “lurking” in email lists. The potential member does not post messages but rather reads messages by others to understand the climate, culture, and relationships between users. At some point, the individual will decide whether or not to enter the open source community. As Miller explains, this can be a difficult point for someone new.

“In order to interpret life in the new organization, the newcomer relies on predispositions, past experience, and the interpretations of others. This phase of socialization can cause a great deal of stress for the newcomer, especially if he or she has ideas about what an organization ‘should’ be like that are contrary to current practices and as a result experiences ‘reality shock’. Thus, the encounter phase encompasses both learning about a new organization and role and letting go of old values, expectations, and behaviors” (pg. 136) ([Miller, 1998](#)).

All of this can be related to how the individual approaches the community and communicates as well as the communication of members already in the community. The final stage of socialization is the metamorphosis stage. In this stage, a user becomes an accepted member of the community. At this point they understand the structure as well as their own role in the community.

When a user is looking to join an open source project, there are a variety of properties of the socialization process that are unique to open source. Van Maanen and Schein believe there are six dimensions of socialization tactics (pg. 139) ([Miller, 1998](#)).

- Collective versus Individual
- Formal versus Informal
- Sequential versus Random
- Fixed versus Variable
- Serial versus Disjunctive
- Investiture versus Divestiture

“Collective socialization refers to taking a group of recruits and putting them through a set of experiences together. Individual socialization refers to the processing of recruits singly in isolation from each other” (pg. 139) ([Miller, 1998](#)). Open source communities are unique as in most cases there is a hybrid form of this dimension used. New participants are certainly socialized individually, however, they have the ability to communicate and see the socialization process of everyone else in the community at the same time through email messages and communication. In the next dimension, open source communities use a process of information socialization in that new participants are not differentiated from other members. However, because the relationships and power structure are already established between members, spotting a new participant is fairly easy. Open source socialization is also quite random in that there are no discrete steps to becoming socialized. This may be different for each user based on what the user wishes to accomplish or contribute. In similar fashion, the timing of socialization

is unknown so there is a significant amount of variability involved. Open source projects are seen as using serial socialization because, “experienced organizational members serve as role models” (pg. 139) (Miller, 1998). The new participant is able to watch the behavior of others as well as ask for help from veterans of the community. Lastly, a form of investiture socialization is prevalent in open source communities because the individual characteristics of the new participant are valued. These skills of communication and technical aptitude are improved through the process of participating in the community.

An individual wishing to join an open source community has a significant number of hurdles to clear before they are even considered a low level developer. There is a gap in the knowledge between new developers and core developers in differing areas such as the community, codebase, and potentially technical knowledge (e.g. programming languages).

These hurdles result in a high barrier to entry for newcomers wishing to join the project. A newcomer must have the necessary technical skills to join the project. They must also read and understand portions of the existing codebase. Given the size and complexity of some open source projects, this in itself can be daunting. The individual must also join and watch the mailing list and communication channels of the project to start to piece together the social power status of the other members and ascertain where the project is going as a whole. At this point, a huge amount of time and energy has been consumed by the individual and they are not even in the project yet. Finally, after coding some small features and successfully persuading other members in the community that these features are important additions, the user will probably get write access to the source code control and be able to submit the code. At this point, the individual has become a low level developer in the project. This is usually how the process is successfully navigated, however, not everyone who wishes to join an open source project does so in a manner that is healthy for the project as a whole.

Many new participants of an open source project just want to give an idea for a feature. If it is a small feature, this is acceptable but some users want to change the entire direction and scope of the project. By not taking the time to understand the long term goals of the project as well as not having any code, their idea is often considered useless by the community at large. This new participant then gets discouraged by the lack of positive response and then leaves.

This can negatively affect the community. It is much better for a new participant to arrive at the group with a solution to a problem or a new feature already coded so that the community can examine the code and determine how it fits within the overall scope of the project.

New participants will surely ask questions but asking the right kind of questions is important. A user asking questions that can be found on the website or that are available in the mailing list archive is wasting the time of the developers. In addition, if a user is arguing over minute details and constantly wanting perfection, it again wastes the time of the developers. The topic in question can also influence the amount of responses that are generated. For a basic feature or general change to the code, everyone in the community can have an opinion because it is so broad. On the other hand, something very specific or technically complex will yield far fewer responses and communication in the community. This means picking appropriate topics to discuss and wording them carefully is important. If the topic is sufficiently broad, be prepared for a large amount of discussion that may waste time and not yield a conclusive answer. Many of these tips have to be learned through experience, but they provide interesting insight into things new entrants should abstain from when joining an open source project. Successful members of a project understand their role in the community, as well as understand that the community as a whole is more important than one individual. Since these relationships and power between members is such a crucial aspect of open source projects, academic research on open source communities over the last few years has concentrated heavily on using a network analysis approach.

2.3.4 Data Mining and Network Analysis

The availability of large information datasets in open source projects such as email lists and source code commits has resulted in many researchers collecting this data and performing network analyses. A network by definition is a collection of nodes and edges. Edges represent the connections between nodes. These edges can be weighted to signify the strength of the connection between the nodes. A network provides common properties that can be used to explain relationships between nodes as well as the dynamics of the network as a whole. Any open source community with more than one member can be mapped as a network. Properties

such as relationships and power between members become very important. Since open source communities are constantly changing, understanding the group dynamics can be helpful not only for researchers but for participants in that community.

“FOSS [free and open source software] project participants self-organize around the expertise, reputation, and accomplishments of core developers, secondary contributors, and tertiary reviewers and other peripheral volunteers. This in turn serves to help create an easily assimilated basis for their collective action in developing FOSS. Thus, there is no assumption of communal or egalitarian authority nor utopian spirit. Instead what can be seen is a pragmatic, continuously negotiated order that tries to minimize the time and effort expended in mitigating decision-making conflicts while encouraging cooperation through reiterated and shared beliefs, values, norms, and other mental models” (Scacchi, 2007).

The structure, climate, culture, and all other properties of an open source project influence this dynamically changing structure so that the best participants and contributions have the most impact on the future of the project.

One network analysis paper explored the relationships between code modules (Gonzalez-Barahona, López, Robles, Sistemas, & Juan Carlos, 2004). By taking snapshots of the network over time, the images showed new modules being created as well as sub-communities forming using colored nodes. The Girvan-Newman algorithm was used which takes large highly connected networks and converts it into a smaller network. This simplification process yields less detailed information but is easier to visualize. Another paper examined developers participating in one or more open source projects. The number of projects per developer is shown to be a power-law distribution (Madey, Freech, & Tynan, 2004). In addition, the researchers use agent-based modeling to examine the system. “Agent-based modeling is a technique for understanding the temporal dynamics and emergent properties of self-organizing system by simulating the behavior of individual components of the network over time” (Madey et al., 2004). In this case, Madey et al. (2004) were using a simulation of agents to represent open source developers working on projects. While fitting this simulation data to the true data

gathered was difficult, the use of agent-based modeling to explore open source communities is an important contribution.

While a lot of research has gone into studying OSS communities, there are still many areas that could be improved. Understanding the communities as well as improving the health of the communities can be refined.

2.4 Current Problems in OSS

The stark reality is that most OSS projects fail. There are hundreds of abandoned projects out there that have ceased development. Whether due to time constraints, lack of interest, or other issues, these projects may just be a byproduct of the open source system as a whole. The constant evolution of projects and contributors means there is a high turnover rate both in projects and community members. Increasing understanding of OSS communities would assist us in determining if this is just part of the system or if it is something that can be addressed and minimized.

Researchers have found that large numbers of new developers motivated to contribute to a project do not gain access to the project because they simply are not allowed into the group. [von Krogh et al. \(2003\)](#) found that of the participants in the open source Freenet community who posted on the mailing list, 10.5% did not receive any reply to their initial posting and subsequently did not appear on the list again. These results do not include the number of individuals who received one or two responses and then never contributed code or left the community. These examples represent a large number of individuals who could benefit from tools or services to help them more rapidly become integrated or developer a greater understanding of the community. In addition, [von Krogh et al. \(2003\)](#) found that there was a substantial amount of time (from a few weeks to several months) spent watching email lists to understand the community before posting. Shortening this time would hasten the addition and socialization of new members and could allow for more members by preventing them from dropping out.

In addition to barriers to entry, another area of OSS that needs significant attention is gender inequality ([Holliger, 2007](#); [Nafus, Leach, & Krieger, 2006](#); [Powell, Hunsinger, & Medlin,](#)

2010). Discrimination, harassment, stereotypes, and few female role models in OSS seem to create a self-perpetuating problem. In one study of 2784 OSS developers, only 1.1% were female (Ghosh, Glott, Krieger, & Robles, 2002). This substantial gender gap is a problem not only for OSS but all science, technology, engineering, and math (STEM) fields. The percentage of women in STEM fields in the US is only 25% (Beede et al., 2011). The percentage of women who graduate in undergraduate Computer Information and Sciences is even lower at around 18% (National Center for Education Statistics, 2015). Other problems such as income disparity also exist. Improving the retention and easing the entry to OSS projects would hopefully increase the number of women who participate. Various groups have tried to rectify this by supporting women in OSS. Support channels such as the Ada Initiative (Ada Initiative, 2015), Girl Develop It (Girl Develop It, 2015), Linux chix (Linux Chix, 2015), Debian Women (Debian Women, 2015), Ubuntu Women (Ubuntu Women, 2015), Drupal chix (Women in Drupal, 2015), and DevChix (Dev Chix, 2015) have all been created to improve this inequality. However, more work needs to be done. While increasing the number of women in OSS is not a direct goal of this work, it is hopeful as an indirect result that can be obtained.

2.5 Existing STC Tools

There are a fair number of tools in both the research and commercial realm that help developers visualize communication patterns and code changes as well as collaborate. Sack et al. (2006) created a tool called the Open Source Browser which assists developers visualize STC. Using this software, they identified trajectories of participation in the Python community that resulted in success by utilizing the tool as well as ethnographic analysis (Ducheneaut, 2005). Sack et al. (2006) tracked a participant who became integrated into the community over a 10 month time frame. Over this time period, this member progressed from peripheral activities such as reporting bugs and submitting patches to more central ones such as taking charge of code artifacts and developing new code. Other tools such as CodeSaw (Gilbert & Karahalios, 2007) and Augur (de Souza, Froehlich, & Dourish, 2005), allow developers to visualize the amount of code commits and communication through time, whereas Tesseract shows missing communication links between developers, source code dependencies, and a timeline (Sarma &

Herbsleb, 2008b; Sarma, Maccherone, Wagstrom, & Herbsleb, 2009); while Ariadne (de Souza, Quirk, Trainer, & Redmiles, 2007) matches similar developers based on the socio-technical dependencies. Ensemble (Xiang et al., 2008) is a recommendation tool that matches developers to artifacts while Palantir (Sarma, Noroozi, & van der Hoek, 2003; Sarma, Redmiles, & van der Hoek, 2008, 2012) and Hipikat (Cubranic & Murphy, 2003; Cubranic, Murphy, Singer, & Booth, 2005) can be used to understand workspace awareness. ProxiScientia can be used for real-time visualization of coordination needs (Borici, Blincoe, Schröter, Valetto, & Damian, 2012). It provides developer-centric and task-centric graphics in an ego-centered visualization. Codebook crawls source code, work items, people, and other information to create a corpus of data (Begel, Phang, & Zimmermann, 2010). This data can then be searched and filtered using regular expressions. Expertise browser (Mockus & Herbsleb, 2002) and expertise recommender (McDonald & Ackerman, 2000) both provide recommendations regarding who to contact given their expertise. By combining source code information and mining who has worked on which section, individuals can be identified who have specific as well as broad knowledge of the project. ALERT is an ambitious upcoming project that seeks to combine data-mining, recommendation systems, visualization, and more into one succinct program (ALERT, 2015). Wolf is a tool that identifies people who are impacted by changes in the codebase (Figueiredo & de Souza, 2012). This is also known as impact analysis. PIVoT (Project Insights and Visualization Toolkit) provides data collection and analysis on topics such as code quality, unit testing effort, development efficiency, code churn, and team analysis (Sharma & Kaulgud, 2012). Syde shows real-time changes in the system and identifies merge conflicts immediately allowing developers to do better planning (Hattori & Lanza, 2010). Tukan defines different modes of collaboration that a developer can be utilizing during code development (Schümmer & Haake, 2001). In this way, it focuses the attention of the developer and prevents unnecessary distractions. In addition, the tool provides tightly-coupled collaboration. IBM's Jazz software is a large project that integrates various plugins into the Eclipse IDE (Frost, 2007). Some of the features Jazz includes are automatic notifications so developers can keep up-to-date, communication and messaging, extensible plugins, and integration of version control systems (Rodríguez, Ebert, & Vizcaino, 2010). ProjectWatcher is a prototype that mines local repository information before

it is checked into a central version control (Schneider, Gutwin, Penner, & Paquette, 2004). The tool shows a visual model of what developers are currently working on in the source code. Lighthouse is an Eclipse plugin that alerts developers immediately to a merge conflict (da Silva, Chen, der Westhuizen, Ripley, & van der Hoek, 2006). FASTDash is a code awareness tool that shows source code activity and information on who is currently looking at specific files (Biehl, Czerwinski, Smith, & Robertson, 2007). CollabVS (Dewan & Hegde, 2007) is a merge conflict resolution system for Visual Studio that informs developers before they commit the code that a conflict has occurred. Boa is a language and framework for querying SVN repositories of large datasets such as Sourceforge and providing counts and other statistical information (Dyer, Nguyen, Rajan, & Nguyen, 2012). Maispion allows for visualizing email communication as well as source code repositories (Stephany, Mens, & Gırba, 2009).

Although these tools are helpful, most are more research focused and have not been tested and evaluated over time in existing communities. In addition, there are significant usability questions surrounding how to present this data effectively. Most of the tools appear to be geared towards only technical code development.

2.6 Contributions

This research has a number of contributions, the details of which are covered in the following chapters. The first contribution is a survey that examined two existing open source communities, KDE and Mozilla. The purpose of the survey was to gauge the potential difference in perceptions between new members and existing members of the community on the topic of socialization. Both groups provided quantitative and qualitative data on their respective community.

The three major academic contributions in this work are in the algorithms used to assess communities and provide information. A processing tool and website called Jamii was developed that analyzes source code and email communication through the use of three new algorithms. The impactful commits algorithm identifies commits that have had a substantial impact to the technical dependency structure of the source code. The developer knowledge algorithm selects developers who have knowledge or expertise on a file based on the commit history. Finally, the

communication suggestions algorithm provides suggestions regarding who a developer should increase communication with on the mailing list based on shared technical dependencies.

Subsequently, the Jamii tool was provided to the KDE and Mozilla communities to use. During this process, a survey and interview identified the utility of the tool for improving the socialization process in these communities. This contribution shows the advantage of using a tool like Jamii to improve the congruence and health of open source communities. While these algorithms and results were tested on open source communities, they could also be applied to other fields where large scale collaboration is needed. The results are fundamental contributions to the field of STC.

CHAPTER 3. SURVEY OF COMMUNITIES

This research focuses on two potentially significant barriers to entry for new developers wishing to join an OSS project that can influence STC and the quality of the project: understanding the source code, and the communication structure of the group. From a code structure standpoint, inexperienced developers may have difficulty grasping the structure of the codebase. With regard to the communication structure, understanding the power structure and general organizational climate of the community as well as understanding who is working on which sections of code that are interdependent with new developers is important. Consistent with the STC perspective, understanding both technical and social processes will lead to the effective development of productive and efficient OSS communities.

3.1 Communities

As part of this research, Mozilla[®] and KDE[®], two open source communities were selected to survey and analyze. They were selected because the barriers to successful socialization are more pronounced in large communities where it becomes increasingly more difficult to know everyone in the community, understand the entire codebase, and keep track of all the current development changes.

Mozilla is most famously known for being the guiding entity behind the web-browser Firefox[®]. Firefox includes 10 million lines of code and receives 3000 commits from 350 unique developer contributors each month¹. The Mozilla Foundation ([Mozilla Foundation, 2015](#)) is the registered nonprofit organization that supports the community and development.

KDE is one of the largest open source communities. KDE is an extremely popular window manager and suite of applications for Linux. KDE has roughly 6 million lines of code,

¹<http://www.ohloh.net/p/firefox>

1800 unique contributor accounts, and 20 new developers who contribute their first code each month². KDE e.V. (KDE e.V., 2015) is the registered nonprofit organization that supports this community.

Meetings with community managers and representatives from Mozilla through video conferencing helped brainstorm what information would be useful in guiding new users. Email exchanges with developers and leaders in KDE communicated their thoughts on the importance of the socialization process. Both communities were receptive to the idea of the website. For Mozilla specifically, one member commented that community management needs to transform from an “art” to a science through the use of data and the creation of tools.

3.2 Survey

A multi-faceted user study examining the two OSS communities gathered quantitative and qualitative self-report data from core developers as well as new/lurking developers. The surveys and analysis were split into two phases. First, an exploratory survey examined the demographics, technical profile, and socialization process through self-report data in an online survey. The results from this survey influenced the development of the Jamii website. After the Jamii website was developed, a final survey and set of interviews were distributed to the two communities to assess the utility of the Jamii website and the potential improvement in the socialization process. The exploratory initial survey is detailed in this chapter while the final survey is presented in chapter five. An outline of the survey process can be seen in Figure 3.1.



Figure 3.1 Survey Development and Order

For the initial survey, core members were defined as developers who have coordinated the development of the project and have made significant contributions (Ye & Kishida, 2003). Consistent with research (Nonnecke & Preece, 2000; Ridings, Gefen, & Arinze, 2006; von Krogh

²<http://www.kde.org/presspage/>

et al., 2003), lurking individuals were defined as those who have registered for the project mailing list but who have not committed any code. In order to gain a comprehensive analysis of core developers and new/lurking developers from a social perspective, this research assessed multiple components through a variety of primarily Likert questions.

Unfortunately, there has been minimal research examining key demographic characteristics of OSS communities. Ghosh et al. (2002), David, Waterman, and Arora (2003), and Ridings et al. (2006) are examples of surveys that examined demographic as well as motivational factors of OSS communities. This survey builds upon the previous research by asking general questions such as member's age, gender identity, technical skills items such as educational and programming experience, previous experience with other communities, and occupation. Items assessing each member's role in the current project, how they first contributed, length of time involved in the community, current role, and motivational aspects of community participation (paid or unpaid) were also included. The primary focus was on subjective assessments of issues/concerns within the project and the community member's perception of the overall socialization process.

3.3 Characteristics of Core Developers and New/Lurking Individuals

Understanding the background and perceptions of a project's core developers and their relationship with the online community is important in understanding the community's culture (Ducheneaut, 2005). This work assessed core developers' perceptions of how they joined the community and how they felt new developers should join the community. Of particular interest were core developers' beliefs about socializing new developers. Specific items of interest included the assessment of perceived barriers of entry for new developers (social, technical, etc.), the community's ability to successfully socialize new developers, and the subsequent ability to integrate new developers into the existing community.

By comparing the viewpoints of core developers in the community and their acceptance of new developers, the survey examined whether both groups have similar perceptions. A difference in perceptions between existing and new developers may suggest a disconnect that may prevent the successful integration of new developers. This social comparison of demographics and beliefs between groups is important because it demonstrates how each group views the

socialization process. Any differences or dissimilarity in opinions and experiences may suggest that the socialization process may be more complex than initially considered and may highlight new barriers not yet considered.

A variety of personality factor measures were available as possibilities to include in the survey. The breakdown of measures and associated questionnaires can be seen in Table 3.1.

Table 3.1 Measures and Associated Questionnaires

Measures	Reference
Agreeableness and Neuroticism	(John & Srivastava, 1999) (The Big Five)
Benevolence in the community	(Ridings et al., 2006)
Transactive memory	(Lewis, 2003)
Investment and commitment	(Rusbult, Martz, & Agnew, 1998)
Functional communication	(Leathers, 1972)
Need for cognition	(Cacioppo et al., 1996)
Willingness to communicate	(McCroskey, 1992)
Need to belong	(Leary et al., 2007)
Hypersensitive narcissism scale	(Hendin & Cheek, 1997)

Transactive memory is the process by which members in a group obtain and store knowledge ([Wegner et al., 1991](#)). Members understand what they know and importantly understand what others know. Transactive memory and the associated scale was chosen because it best measures the expertise and understanding of members which could be directly improved through the development of the Jamii tool and website.

3.4 Survey Results

The survey ran for just over a month from June 21, 2013 to August 26, 2013. The survey questions can be seen in Appendix A. All analysis was done using the R software version 3.1.1 and supporting packages ([R Project, 2015](#)). After removing empty submissions, there were a total of 42 responses across Mozilla ($n = 19$) and KDE ($n = 23$). The Mozilla respondents self-reported gender as 15 male and 3 female. Similarly, the KDE gender demographics were 20 male and 1 female. No participants selected the fill-in-the-blank entry for “other identification” or “prefer not to specify” for gender. Each community was analyzed separately and no statistical comparison was made to examine the differences in responses between the two communities.

The country distribution showed that over half of the participants were from the United States and Europe (see Table 3.2). GeoIP lookups on the IP address participants used to complete the survey matched the countries that they reported. The Mozilla Foundation is based in the United States while KDE e.V. is based in Germany. This is reflected in the distribution of countries in Table 3.2.

Table 3.2 Country Distribution

Country	Mozilla	KDE	Total
Austria	2	0	2
Brazil	1	0	1
Canada	2	0	2
Switzerland	0	1	1
Czech Republic	0	1	1
Germany	3	4	7
Spain	0	2	2
France	1	1	2
United Kingdom	0	2	2
Greece	0	1	1
India	0	1	1
Italy	0	1	1
Netherlands	1	0	1
Romania	0	1	1
Russia	0	1	1
Turkey	0	1	1
Taiwan	1	1	2
United States	7	3	10

Many participants identified as being bilingual in both Mozilla ($n = 18, M = 2.05, SD = 0.72$) and KDE ($n = 21, M = 2.42, SD = 1.02$) as can be seen in in Figure 3.2. English was most often listed as a first or second language. When asked about their highest level of schooling or education, the survey showed that half of participants have a bachelor’s degree (see Table 3.3). The amount of time in years participants have spent in Mozilla ($n = 14, M = 5.61, SD = 4.91$) and KDE ($n = 15, M = 4.72, SD = 3.44$) hovers around approximately 5 years (see Figure 3.3).

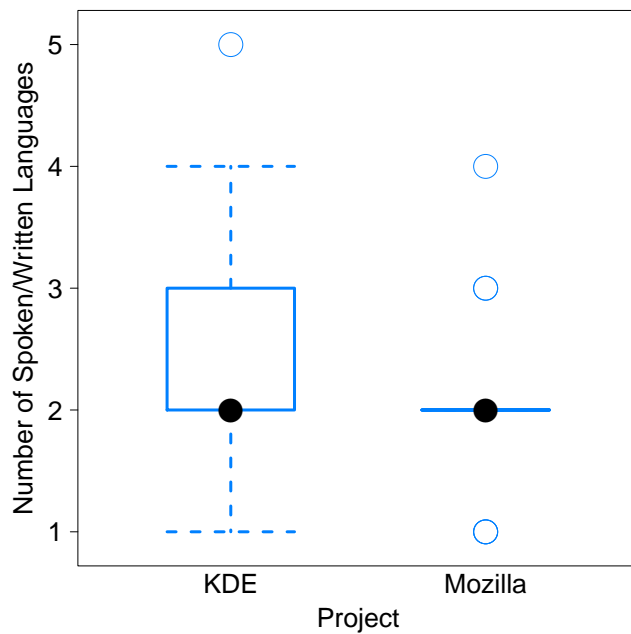


Figure 3.2 Spoken/Written Languages

Table 3.3 Highest Education/Schooling

Education Level	Mozilla	KDE	Total
High School/Secondary School	1	4	5
Undergraduate/Post-Secondary	8	11	19
Professional Degree	1	1	2
Master's Degree	5	4	9
Doctorate Degree	2	1	3

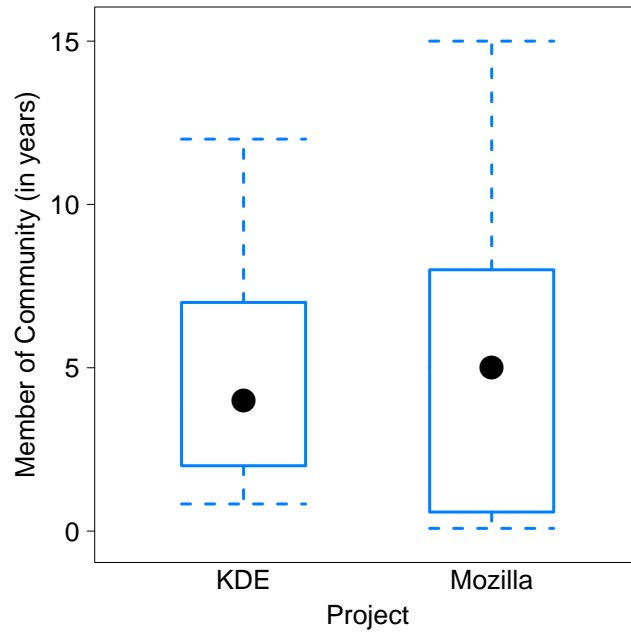


Figure 3.3 Number of Years in Community

3.4.1 Overall Responses

Experience contributing to FLOSS communities was measured using a Likert question (1=None, 5=A Lot). The results for Mozilla ($n = 17, M = 3.76, SD = 1.30$) and KDE ($n = 19, M = 3.73, SD = 0.99$) were similar and can be seen in Figure 3.4.

Experience in computer programming relative to the average Open Source contributor was measured using a Likert question (1=None, 5=A Lot). The results for Mozilla ($n = 17, M = 3.35, SD = 1.16$) and KDE ($n = 19, M = 3.10, SD = 0.80$) show participants felt they have a roughly average level of computer programming experience in both communities. These results can be seen in Figure 3.5.

Participants noted current involvement in approximately two FLOSS communities: Mozilla ($n = 17, M = 1.76, SD = 2.25$) and KDE ($n = 18, M = 2.11, SD = 1.23$).

When asked how they spend their time in each of the respective communities each week, participants responded that much of the time was spent communicating and programming (see Table 3.4). One participant entry was removed because the number of minutes entered was more than possible for one week. One question asked participants how long they spend

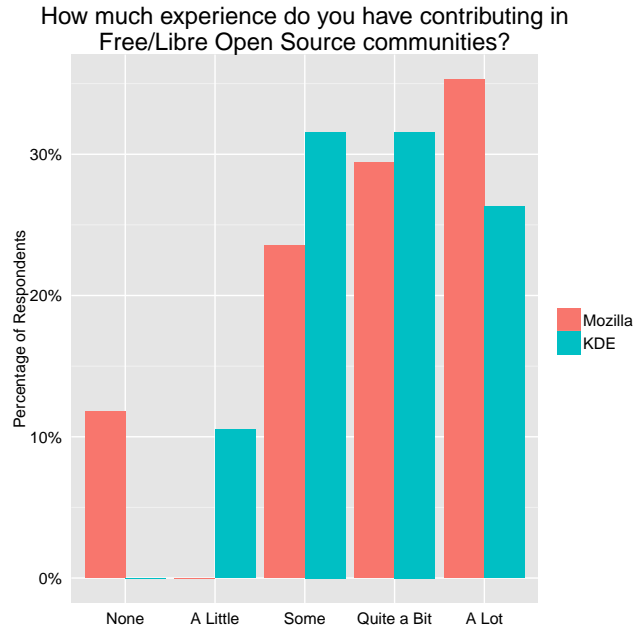


Figure 3.4 Experience Contributing to FLOSS

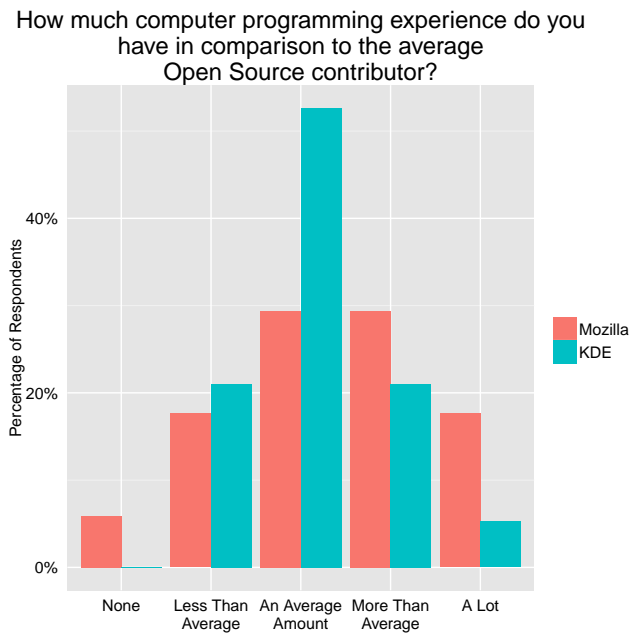


Figure 3.5 Computer Programming Experience

responding to existing bug reports. For KDE specifically, this had a particularly high standard deviation due to an extreme outlier. This participant spends 2000 minutes a week (roughly 75% of their time) on bug reports.

Table 3.4 Time Spent in Community Each Week

		Mozilla ($n = 15$)		KDE ($n = 17$)	
		Time in minutes M (SD)	Relative Percentage	Time in minutes M (SD)	Relative Percentage
Programming	New Features	164 (267.20)	13%	107.94 (193.70)	16%
Programming	to Fix Bugs	171 (282.11)	13%	87.64 (222.50)	13%
Filing	New Bug Reports	74.66 (110.94)	6%	16.58 (27.47)	3%
Responding to	Existing Bug Reports	135 (183.43)	11%	122.47 (484.05)	19%
Reading Code	Written By Others	149 (255.54)	12%	86.47 (156.27)	13%
Writing	Documentation	31.67 (41.48)	3%	15.58 (48.18)	2%
Communication	(email, IRC, etc.)	406.66 (419.82)	32%	171.76 (211.99)	26%
Other		133.33 (288.63)	10%	49.11 (132.43)	8%
Total		1265.33 (1012.18)	100%	657.6 (854.9)	100%

The time spent on “other” activities also had a high standard deviation for both Mozilla and KDE. One explanation for this is participants used this category to indicate work in localization or other areas. Localization is the process of translating text from one language to another. This is important in ensuring applications can be understood by users all over the world. Some of the respondents spent 50% or more of their time in these “other” activities indicating contributions to areas outside programming.

Not all people who contribute to open source projects are volunteers. Some receive monetary compensation for their efforts. In this study, 65% of the Mozilla participants indicated they receive monetary compensation for their open source efforts; whereas, almost 90% of the KDE

participants were unpaid volunteers (Table 3.5). This reflects a larger grass-roots influence in the KDE community as compared to Mozilla.

Table 3.5 Paid Compensation (Employees)

	Unpaid	Paid
Mozilla	6	11
KDE	17	2

Quite a few participants in the survey self-identified as being at a relatively beginner level in their respective communities. The Likert question assessed both their current level as well as what level they want to obtain (1=An Outsider, 7=A Core Member) (see Figure 3.6). A high number of participants in both Mozilla and KDE indicated they wanted to be substantially involved in the community at a core level.

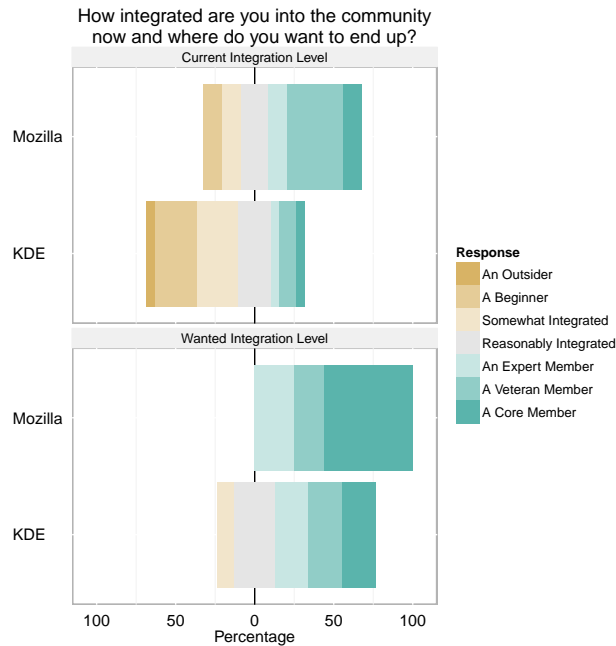


Figure 3.6 Integration Level: Current and Future

When examining the difference between their wanted integration level in the community and their current integration level, all participants in both communities wanted to either stay at their current integration level or increase. The results for Mozilla ($n = 17$, $M = 1.35$, $SD = 1.22$) and KDE ($n = 23$, $M = 1.39$, $SD = 1.67$) were similar and can be seen in Figure 3.7.

In general, participants wanted to increase their integration by one or two levels in the Likert response categories. This indicates a substantial number wanted to increase their participation and level of involvement.

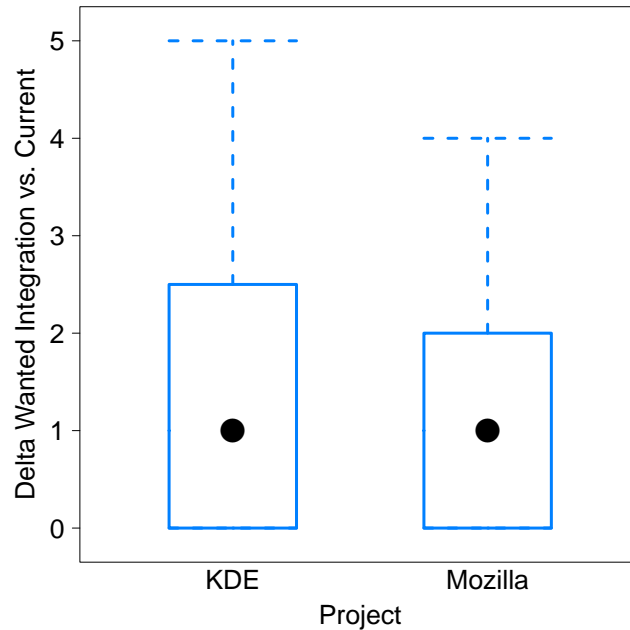


Figure 3.7 Delta Difference: Wanted Integration vs. Current Integration

When asked how many days they spent watching and reading the mailing list before posting, there was a wide range of responses for both Mozilla ($n = 12$, $M = 304.91$, $Mdn = 39.5$, $SD = 829.76$) and KDE ($n = 17$, $M = 319.70$, $Mdn = 60$, $SD = 649.20$).

Participants were asked via a Likert scale how easy it is for a new member to join the community (1=Very Difficult, 7=Very Easy). Responses for Mozilla indicate participants feel it is more difficult as compared to KDE (see Figure 3.8).

Most participants in KDE feel new members are quickly integrated into the community. For Mozilla approximately 50% felt new members are not quickly integrated (see Figure 3.9). Both Mozilla and KDE participants felt the community treats new members with respect (see Figure 3.10). These responses were gathered via a Likert scale (1=Strongly Disagree, 7=Strongly Agree).

How easy is it for new members to join the community?

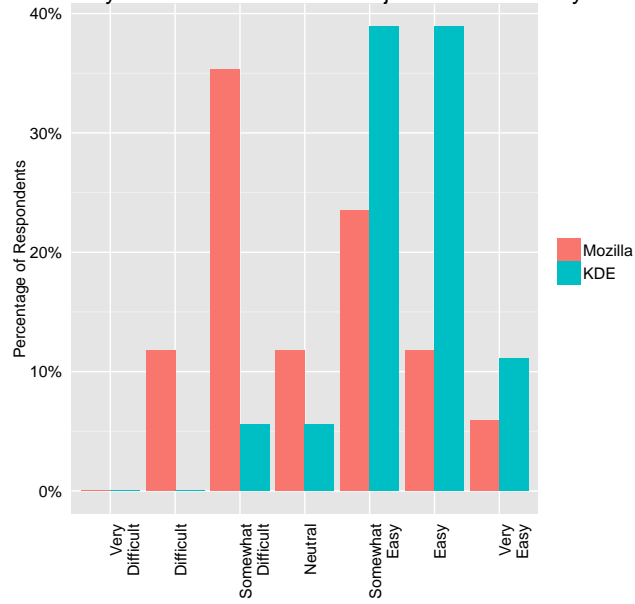


Figure 3.8 New Members Join

New members are quickly and easily integrated into the community.

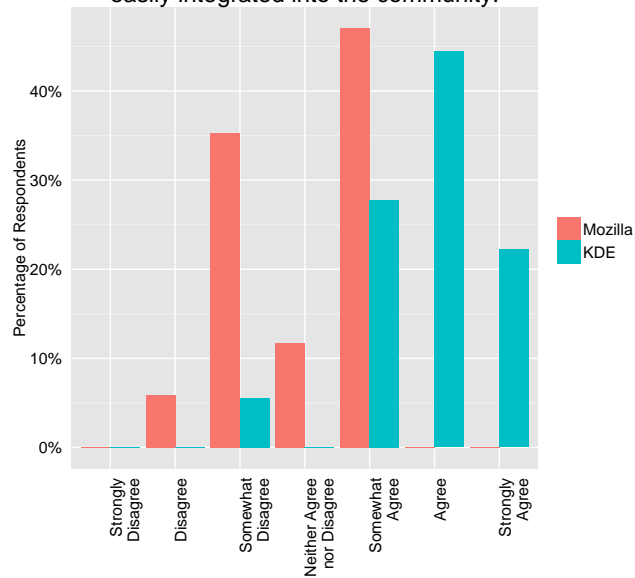


Figure 3.9 New Member Integration

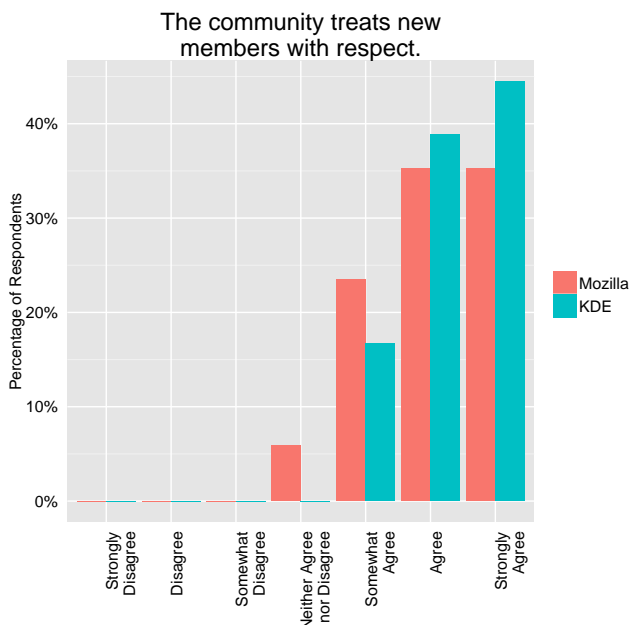


Figure 3.10 New Member Respect

Participants rated social communication as one of the more important skills that new members should possess when joining the community. This was gathered via a Likert scale (1=Not at all important, 7=Extremely Important) (see Figure 3.11).

Participants were asked via a Likert scale (1=Very Ineffective, 7=Very Effective) how well they were able to track various processes as part of the community (see Figure 3.12).

Transactive memory was measured using the Transactive Memory System Scale which asks questions on a Likert scale (1=Strongly Disagree, 5=Strongly Agree) (Lewis, 2003). Two questions were selected from each of the three categories, specialization, credibility, and coordination. These two questions were averaged into a single latent variable score, the results of which can be seen in Figure 3.13. For the most part, participants felt that community members had specialized knowledge in different areas of the project and that the members were knowledgeable and confident in their abilities. There was a little more ambivalence regarding coordination however. In particular for Mozilla, some participants disagreed or were unsure about how the community accomplishes tasks and how they are completed.

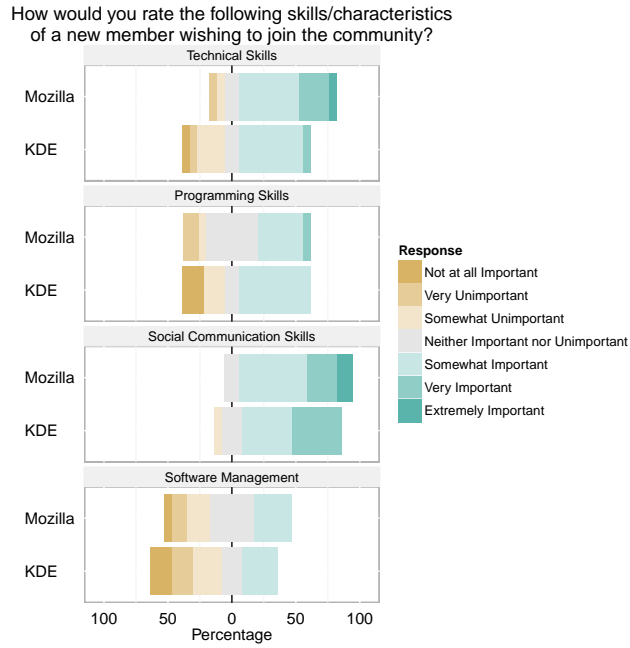


Figure 3.11 Rating of Skills for New Members

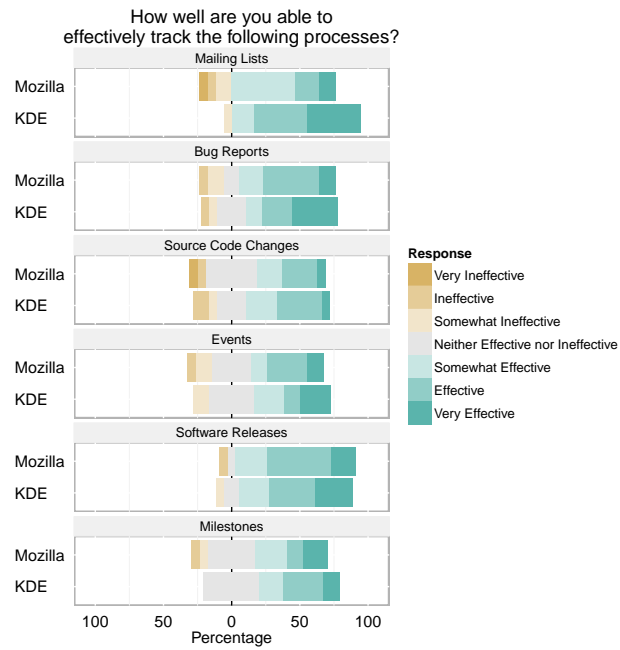


Figure 3.12 Keep Track Processes

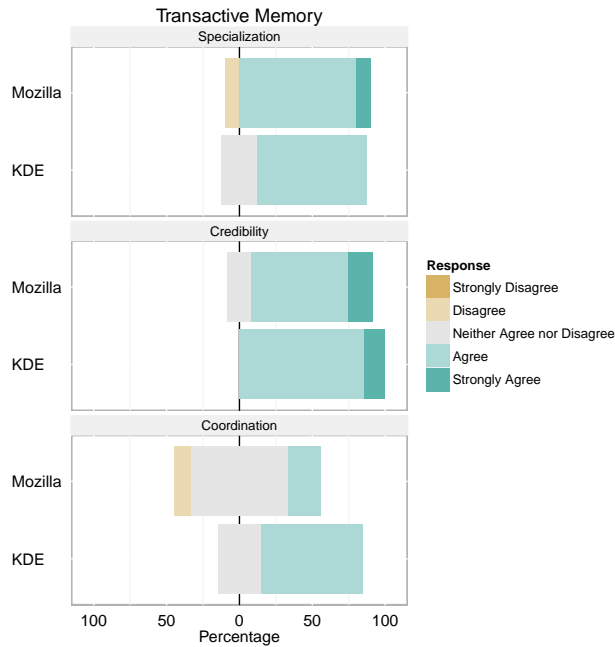


Figure 3.13 Transactive Memory

3.4.2 Qualitative Results

Qualitative results as expressed by open-ended written comments from participants paint a predominantly positive picture of the two communities. When asked, “are there ways the community can support new/potential members in their desire to join?”, participants suggested increased personalized outreach, interaction, and mentoring. In addition, technical challenges such as understanding the build process and version control were mentioned. While no single comment theme emerged in this question, the variety of different suggestions indicates that the socialization process is a complicated and multi-faceted process. All comments were corrected for small spelling mistakes. The comments for Mozilla can be seen in Table 3.6 and the comments for KDE can be seen in Table 3.7.

An additional question attempted to solicit what kinds of new features or functionality tools could assist in the contribution process. This question was primarily aimed at attempting to identify features that could be developed as part of Jamii. The question asked, “what tools or functionality (new or existing) would help you contribute?”. Participants indicated existing tools such as development applications, bug tracking, and localization. For possible future tools

Table 3.6 Qualitative Comments Mozilla

Are there ways the community can support new/potential members in their desire to join?

- “yes- onboarding and enculturation and exposure to ‘how things get done‘ plus most common ways of being open for people new to open culture who are non-technical.”
- “mozilla.org/contribute, local community portal site, mozilla reps everywhere.”
- “Better documentation. Better outreach. Better follow-through. Better follow-up. More personalized interactions.”
- “Find better ways to match new contributors to areas of interest. <http://www.whatcanidoformozilla.org> is a good first step, but misses many areas of the project.”
- “we have a couple websites to make it easy to find something to work on <http://www.whatcanidoformozilla.org>, <http://www.joshmatthews.net/bugsayoy>”
- “Learning the code base is extremely challenging. I think there needs to be people whose sole task is to help new people on the team. More documentation describing key areas would be very helpful. The documentation is severely lacking, though firefox is an extremely fast moving code base, so it’s extremely difficult to keep documentation up to date. What would be nice is to have some individuals assigned as go to people for specific modules.”
- “For programming members, pull/build process is really the biggest hurdle I think. I wonder if there are ways we could simplify that process. Maybe help users who just want to contribute javascript/java code skip the libxul building process entirely.”
- “Easy to solve bugs, welcome tasks, Onboarding package”
- “If there would be open issue lists for each mozilla group, then, new members can easily find [an] area to help. I noticed one case where such issue lists spread to multiple sites, which might seem difficult for new members to understand which site is the latest and/or helps from new people would be helpful.”

Table 3.7 Qualitative Comments KDE

Are there ways the community can support new/potential members in their desire to join?

- “We fail to turn people who are not proactive into contributors (might also be a good thing because we keep a proactive working culture this way). We could also do better at attracting non-programmers like artist/designers. I have no idea how to solve this.”
- “The documentation needs to be up to date. KDE is quite good in that with the beginner’s manual.”
- “Make more beginner tutorials/junior tasks and make those that already exist easier to find.”
- “Mentoring newcomers should be better organized”
- “More actively advertising junior jobs and having more people able and willing to mentor the new members”
- “Video tutorials and interactive courses.”
- “There is documentation on the wiki, and on mailing lists which I read, there is generally a helpful response to new people.”

or functionality, there was a wide range of responses such as finding expertise of members for non-technical areas, better review tools for source code, and changes to bug tracking software. The comments for Mozilla can be seen in Table 3.8 and the comments for KDE can be seen in Table 3.9.

Table 3.8 Qualitative Comments Mozilla

What tools or functionality (new or existing) would help you contribute?
<ul style="list-style-type: none"> • “Ways to find specific expertise in other members- not just in category but something more granular. A tool that tells me that someone is really good at curriculum design and facilitation. This would be especially good for non-technical functional areas that do not live in Bugzilla.” • “reps portal sute, reps.mozilla.org, community region mailing list” • “My tool needs as a long-term contributor are much different than those that new contributors would need, but in general, the recent improvements to bugzilla to make it easier to file and work with bugs. On the technical side: tools to automate the landing of patches from bugzilla, possibly with the ability to uplift those patches to different branches. Automatic bisection tools for failed builds” • “Better review tools / easier to grasp build system (this is being worked on) / maybe more documentation, depending on the area” • “One thing that would be extremely nice is a script that goes through patches and picks lines that clash with the coding style.” • “More time!”

The final qualitative question seemed to show the most stark difference between the two communities. The question asked participants, “what were your experiences in joining?”. KDE participants in particular seemed to express a high amount of intrinsic motivation as opposed to extrinsic motivation which was consistent with other FLOSS communities (Lakhani & Wolf, 2003). All of the responses for KDE indicated a warm and welcoming community when joining. Mozilla on the other hand seemed to have comments that were not quite as positive and showed the challenges of the size and scale of Mozilla’s large community. Difficulty with understanding the codebase and difficulty with understanding what was going on within the community were

Table 3.9 Qualitative Comments KDE

What tools or functionality (new or existing) would help you contribute?
<ul style="list-style-type: none"> • “Our bugtracker/ticket system could be better. I really like the way redmine can associate tickets (bugs and features) with milestones/releases and how it displays them and how it is very easy to build your own todo queue out of it, I would love to see our bugzilla offer a similar feature or a switch to redmine. The actual impact on productivity would probably not justify the work it takes to switch to a different bugtracker.” • “Lokalize / svn” • “better bug tracking software (client side software)” • “I guess any improvements to Lokalize may speed up my contributions as a translator.” • “More developer tutorials are always helpful.” • “IRC, mailing lists / new: a mentoring group would be nice” • “Kdevelop, Kompare, git, Konversation, Kate, Emacs, KDE Techbase, Mail list” • “Something that would make git easier to use (especially when doing anything at all different from the usual) would probably help people to cope with it better. Its lack of user friendliness may well put some contributors off.”

both mentioned by Mozilla participants. The comments for Mozilla can be seen in Table 3.10 and the comments for KDE can be seen in Table 3.11.

Table 3.10 Qualitative Comments Mozilla

What were your experiences in joining?
<ul style="list-style-type: none"> • “Mozilla was a consulting client of mine who turned into a full time role.” • “I was thrown in the deep end, but it worked out ok.” • “*Much* smaller when I joined, still possible to know everyone and keep up with what was going on. New community members (and old ones too) really need to pick and choose where to focus now because there’s just too much going on now.” • “Smooth, since I was an intern” • “It’s extremely challenging at first. The code is difficult to pick up, not because of its quality - it’s some of the best code I’ve read in my life, but because it’s just complex. I live on the east coast, so sometimes it’s challenging to find people to help out on irc. The most difficult part at first is figuring out how to find out how to do things the right way. IRC is the key, but also MXR and searching the code base are helpful too. The challenge there is knowing what to search for. Usually, I ask on IRC what I need to search for to accomplish my task, then use MXR + grep + find to find what I’m looking for.” • “I tried to help out from outside for years and always felt a bit daunted/scared. My experiences asking questions in IRC never went well. Maintaining a build environment on a less than stellar PC was hard. I mostly just wrote addons and prototypes. Lurked bugzilla. Being hired (obviously) made it all much easier.” • “It’s a ‘do-ocracy’.”

3.4.3 Comparing New and Veteran Developers

For all subsequent Likert data, frequency count data, as well as some measures of central tendency are reported. There is a fair amount of disagreement in the statistics community about how to report and analyze Likert responses. Therefore, results from single Likert items utilize Fisher’s exact test. Fisher’s exact test is used when one or more of the cells in the resulting table of responses has a value less than five. When possible, Likert items were combined into a

Table 3.11 Qualitative Comments KDE

What were your experiences in joining?
<ul style="list-style-type: none">• “More than awesome. I felt welcome from the start. Everything went better than I imagined. / It had a positive impact on my life, I simply feel happier and fulfilled.”• “Welcoming”• “Not difficult.”• “It was very easy to join KDE - one just becomes a member without noticing it.”• “Excellent community feedback and quick induction into the community.”• “I switched to KDE from GNOME after being fed up of the latter design choices and found that not only the software was great, but the community was friendly too.”• “Quite welcoming, nobody expected me to be a pro”• “It feels great being a part of this community, even tiny. I only wish I had the time to contribute more.”• “Very warm welcome for anyone willing to contribute.”• “I asked a technical question on a mailing list, and following that I was taken by surprise to be asked to contribute my application into the KDE repository.”• “Very nice and welcoming community.”

single score and analyzed using the parametric Welch Two Sample t-test which does not require equal sample sizes.

In order to compare differences in experience, participants were split into two groups, beginner and veteran, based on their response to the question: “How integrated are you into the community?”. All participants who selected, “an outsider”, “a beginner”, or “somewhat integrated” were placed in the beginner category (Mozilla $n = 4$, KDE $n = 11$). All others were placed in the veteran category (Mozilla $n = 13$, KDE $n = 8$). The number of years spent in the community between beginners (Mozilla $n = 4$, $M = 0.26$, $SD = 0.23$, KDE $n = 7$, $M = 2.83$, $SD = 2.5$) and veterans (Mozilla $n = 10$, $M = 7.75$, $SD = 4.14$, KDE $n = 8$, $M = 6.37$, $SD = 3.42$) shows a large difference and supports the splitting of participants into these two groups.

3.4.3.1 Skills and Characteristics of New Developers

One set of questions asked participants how they would rate various skills and characteristics of new members wishing to join the community on a 7 level Likert scale from, “not at all important” to “extremely important”. Based on correlation results as seen in Table 3.12 the technical, programming, and software management skills are all fairly correlated. Given this, they were averaged into a single numeric value representing general technical skills.

Table 3.12 Correlation Table for Skills/Characteristics for New Members

	Technical Skills	Programming Skills	Social Communication Skills
Technical Skills			
Programming Skills	0.72***		
Social Communication Skills	0.05	0.12	
Software Management	0.44**	0.45**	0.35*

* $p < .05$, ** $p < .01$, *** $p < .001$

This technical skills average was analyzed using a Welch Two Sample t-test. For Mozilla, beginners rated technical skills slightly lower ($n = 4$, $M = 4.08$, $SD = 1.25$) as compared to veterans ($n = 13$, $M = 4.33$, $SD = 0.75$). There was no significant difference between beginner and veteran developers for Mozilla on the rating of needed technical skills, $t(3.69) = -0.37$, p

= 0.72. For KDE, beginners rated technical skills higher ($n = 10, M = 4.23, SD = 0.99$) than veterans ($n = 8, M = 3.12, SD = 1.35$). There was no significant difference between beginner and veteran developers for KDE on the rating of needed technical skills, $t(12.51) = 1.93, p = 0.07$. However, this was quite close to being statistically significant.

3.4.3.2 New Member Joining

Three questions asked participants about the socialization process and joining the community on a 7 level Likert scale. The questions are listed below.

- “How easy is it for new members to join the community?”
- “New members are quickly and easily integrated into the community.”
- “The community treats new members with respect.”

Based on correlation results as seen in Table 3.13 the joining the community and easy integration questions were correlated. Given this, they were averaged into a single numeric value representing the socialization process.

Table 3.13 Correlation Table for Member Joining and Socialization

	Join Easy	Quick Integration
Join Easy		
Quick Integration	0.74***	
Respect	0.10	0.37*

* $p < .05$, ** $p < .01$, *** $p < .001$

This socialization score average was analyzed using a Welch Two Sample t-test. For Mozilla, beginners rated the socialization score slightly higher ($n = 4, M = 4.12, SD = 1.43$) as compared to veterans ($n = 13, M = 4.00, SD = 1.11$). There was no significant difference between beginner and veteran developers for Mozilla on the rating of socialization, $t(4.18) = 0.15, p = 0.88$. For KDE, beginners rated the socialization score slightly higher ($n = 10, M = 5.75, SD = 0.75$) than veterans ($n = 8, M = 5.43, SD = 1.08$). There was

no significant difference between beginner and veteran developers for KDE on the rating of socialization, $t(12.07) = 0.69$, $p = 0.50$.

3.4.3.3 Member Respect

The question asking participants if they feel the community treats members with respect was analyzed as an individual item. For Mozilla, Fisher's exact test did not show a significant difference between beginners ($n = 4$, $M = 6.50$, $SD = 0.57$) and veterans ($n = 13$, $M = 5.84$, $SD = 0.98$), $p = 0.69$. For KDE, Fisher's exact test did show a significant difference between beginners ($n = 10$, $M = 6.60$, $SD = 0.69$) and veterans ($n = 8$, $M = 5.87$, $SD = 0.64$), $p = 0.04$. Interestingly, in both cases, beginners rated the respect level as higher than veterans.

3.4.3.4 Transactive Memory

The transactive memory questions were on a 5 level Likert scale (1=Strongly Disagree, 5=Strongly Agree) (Lewis, 2003). Two questions were selected from each of the three categories, specialization, credibility, and coordination as seen below.

- Specialization
 - “TS1: Each community member has specialized knowledge of some aspect of the project.”
 - “TS2: I know which community members have expertise in specific areas.”
- Credibility
 - “TS3: I trust other community members' knowledge about the project is credible.”
 - “TS4: I am confident relying on the information that other community members bring to the discussion.”
- Coordination
 - “TS5: The community accomplishes tasks smoothly and efficiently.”
 - “TS6: There is a lot of confusion about how the community accomplishes tasks.”

The last question, TS6 was inverted in the coding. The transactive memory data surprisingly was not very correlated as can be seen in Table 3.14. For each of the three categories, the paired questions were averaged into a single score.

Table 3.14 Correlation Table for Transactive Memory

	TS1	TS2	TS3	TS4	TS5
TS1					
TS2	0.14				
TS3	0.14	0.20			
TS4	-0.07	0.20	0.40*		
TS5	0.09	0.19	0.25	0.30	
TS6	0.09	-0.07	0.00	0.21	0.28

* $p < .05$, ** $p < .01$, *** $p < .001$

For specialization, there was no significant difference between Mozilla beginners ($n = 4, M = 3.00, SD = 0.91$) as compared to veterans ($n = 13, M = 3.88, SD = 0.58$), $t(3.78) = -1.82, p = 0.14$. For KDE, there was no significant difference between beginners ($n = 10, M = 3.85, SD = 0.41$) and veterans ($n = 8, M = 3.81, SD = 0.59$), $t(12.03) = 0.15, p = 0.88$.

For credibility, there was no significant difference between Mozilla beginners ($n = 4, M = 4.12, SD = 0.85$) as compared to veterans ($n = 13, M = 4.07, SD = 0.49$), $t(3.63) = 0.10, p = 0.92$. For KDE, there was no significant difference between beginners ($n = 10, M = 4.30, SD = 0.42$) and veterans ($n = 8, M = 4.12, SD = 0.23$), $t(14.42) = 1.11, p = 0.28$.

Lastly for coordination, there was no significant difference between Mozilla beginners ($n = 4, M = 3.62, SD = 0.47$) as compared to veterans ($n = 13, M = 2.96, SD = 0.47$), $t(4.99) = 2.42, p = 0.05$. However, it was very close to significant. For KDE, there was a significant difference between beginners ($n = 10, M = 3.95, SD = 0.36$) and veterans ($n = 8, M = 3.43, SD = 0.41$), $t(14.18) = 2.72, p = 0.01$ with beginners rating the coordination process as smoother than veterans.

3.5 Conclusion

The gender imbalance results match previous work in identifying a gap in the number of women to men in FLOSS communities (Ghosh et al., 2002). This is certainly an area for

improvement and hopefully can be examined through further study of the socialization process. Worldwide distribution shows that over half of the respondents were from the United States and Europe. These results could be skewed slightly since the solicitation for participation and the survey itself were in English. For those not fluent in reading English they may not have been able to respond to the survey.

Most of the participants in both communities have a reasonably high level of experience contributing to FLOSS communities and possess technical expertise in programming. The amount of time spent each week in various areas such as programming, fixing bugs, communication, and other areas indicates that in general for those who responded, people in Mozilla spend more total time than KDE. This may be reflected by the fact that there were a large number of participants responding that receive monetary compensation to work at Mozilla as opposed to KDE. When examining the relative percentages of time spent in various activities, they are overall roughly similar. Communication takes up a large amount of time for Mozilla participants and to a lesser extent KDE participants. The amount of time participants spend fixing and responding to bugs is consistent with the maturity and stability of both Mozilla and KDE projects.

Many of the participants self-identified as being at the beginner stages of integration into the community for KDE and to a lesser extent Mozilla. The integration level that participants would like to have in the future showed that a large number of participants in both communities would like to be expert and core members of the community. This indicates that these participants do not want to be at the edge of the community but rather be active and high level members after their socialization. Understanding wanted integration levels might help project members focus attention and resource on the appropriate people.

The large variance in the amount of time participants spent watching the mailing list before posting matches previous research by [von Krogh et al. \(2003\)](#). This emphasizes the importance of understanding the social fabric of a community before engaging with it.

In the survey areas related to gauging new members and the socialization process, approximately 50% of respondents for Mozilla indicated some level of difficulty in how easy it is for new members to join the community. This suggests a potential area for improvement. The actual

skills that participants felt new members should possess again show some importance for technical and programming skills however also suggests a high importance in social communication skills. For Mozilla, approximately half of the participants disagreed and felt new members are not easily integrated into the community echoing the results of the previous Likert question. In both communities, participants felt new members are treated with respect.

Participants in both Mozilla and KDE felt that keeping track of various processes in the community such as mailing lists, bug reports, source code changes, and other areas were fairly effective. However, roughly one quarter felt that there could be some improvement in many of these areas. Tools focusing on these particular areas could help in these processes, particularly for new members.

The results from the quantitative and qualitative data seem to paint a primarily positive picture of socialization and inclusion for KDE and perhaps a slightly larger, more complex process of socialization and inclusion for Mozilla because of the daunting size of its community and codebase. Interestingly, in many cases for both communities, the ratings for the community on a number of factors relating to the socialization process and the ability for the community to effectively coordinate tasks were actually higher for the beginner participants as compared to the veterans. This indicates that perhaps the beginners have a slightly rosier view or outlook on the community while those that are veterans know the “true” state of the community. This suggests that new participants may have an over-inflated view of the overall ease of joining, participating, and contributing to the community than veteran members. Ensuring realistic expectations for beginners, while at the same time keeping tasks simple to ease the transition process, may be an important step in this socialization process to ensure new members don’t leave the community early.

The results of this study are subject to the following threats to validity. External validity is defined as how well the findings generalize to other populations or groups. In this case, there appeared to be some overall differences in perception between Mozilla and KDE. This suggests the skewed difference in perception between beginner and veteran community members perhaps may not generalize to other large FLOSS projects. Obviously, projects vary in terms of how they’re organized and developed, especially as their size increases.

The survey data was gathered via convenience sampling. Not all members of the community participated. The survey may have pulled participants from more technical backgrounds and missed people in other areas of the community such as localization, advocacy, writing, and other areas. A larger sample size pulling participants from more areas of the community would address this issue. In general, the small sample size obtained for each of the two communities means these results should be taken with that under consideration. Increased participation in the survey would increase the validity, however overall, the trends of the data suggest a predominantly positive health of both communities with potential for improvement in socialization of new participants. The results of this initial survey and ideas suggested were considered and evaluated during the brainstorming and development of the algorithms and Jamii website.

CHAPTER 4. ALGORITHM AND WEBSITE DEVELOPMENT

Overview

In efforts to provide relevant information and analysis to open source project contributors, an online website was developed called Jamii. The name “Jamii” comes from Swahili meaning “community”. Leveraging a distributed version control system Git ([Git, 2015](#)) as well as an email mailing list, the data is parsed and processed. The website has three sections: *impactful commits*, *developer knowledge*, and *communication suggestions*. The specifics of the development of the Jamii website and description of the details of the three algorithms are covered in the following chapter.

4.1 Jamii Online Website

The Jamii website has both a backend processing side as well as a user-facing frontend (see [Figure 4.1](#)). The source code of Jamii is available on the Github website ([Jamii Github, 2015](#)).

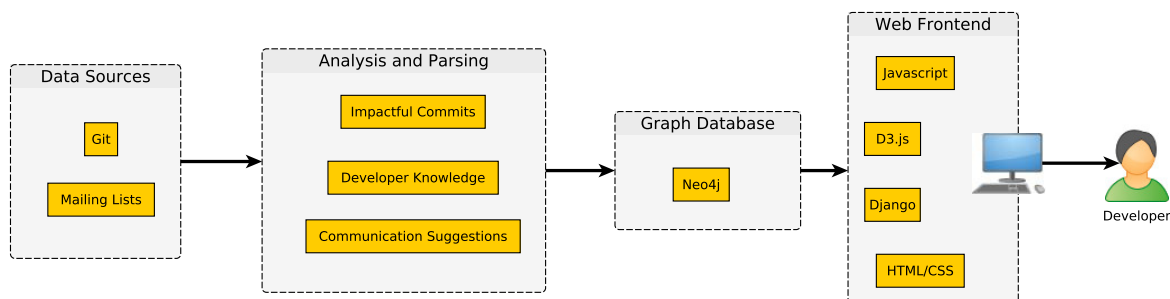


Figure 4.1 Data Mining Sources

Jamii was built on top of a wide variety of libraries and software. All of the backend processing as well as the frontend use Python (Python, 2015) as the programming language. Jamii processes email mailing lists exported to flat text-files in Mbox format as well as source code version control stored in Git (Git, 2015). Identifying technical dependencies between source code files are limited to the C++ language. This is accomplished using the Python Clang bindings in LibClang as part of the Clang project (Clang, 2015). Clang is a compiler and set of tools which is part of the overarching LLVM project (LLVM, 2015). Once the data is processed through the backend, it is stored in a Neo4j (Neo4j, 2015) graph database.

Graphs are a collection of nodes and edges (also referred to as vertices). An edge connects two nodes and can either be directed meaning there is a directionality between the two nodes and a connecting edge or it can be undirected. An example of a directed graph can be seen in Figure 4.2.

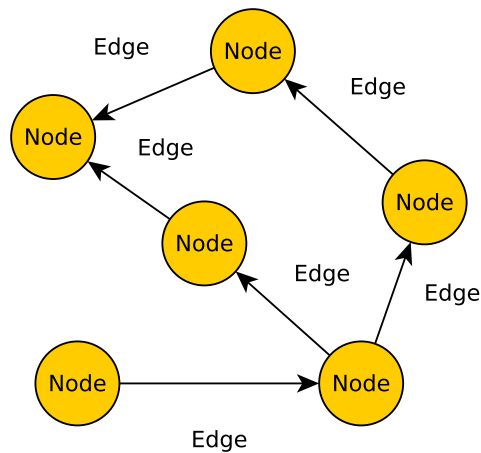


Figure 4.2 Example Graph with Nodes and Edges

Graph databases are relatively new in the realm of databases and data storage. They are categorized as so-called “Not Only SQL” or NoSQL databases. NoSQL databases contrast with more traditional relational SQL databases that use tables to store the information. Querying of the data uses some form of SQL. An example reading the entirety of a specific table in a relational database is as follows:

```
SELECT * FROM tablename
```

Tables can be combined and filtered in a wide variety of ways using SQL queries. One disadvantage of relational databases is that as the number of tables increase the complexity of SQL statements grows. NoSQL databases often have simpler storage of information resulting in potentially faster queries than relational databases. Key-value storage and document stores are two of the more popular types of NoSQL databases. Titan ([Titan, 2015](#)), MongoDB ([MongoDB, 2015](#)), and many other NoSQL databases were compared as possibilities for data storage. The graph database Neo4j was selected over other NoSQL databases and relational databases because it fits the overall use of graphs as data in the STC framework. Neo4j uses nodes and edges with both being able to have named properties with values.

Neo4j uses its own query language called Cypher ([Neo4j Cypher Refcard, 2015](#)) which is similar in concept and functionality to SQL queries. An example Cypher statement that queries all nodes and returns them if they have a property set to 1 can be seen below:

```
start a=node(*) where a.property = '1' return a;
```

Through the backend processing stages; nodes, edges, and properties are added and removed. The final database structure that is queried and returns information to the users can be seen in [Figure 4.3](#). The squares are nodes and the edges are arrows with example properties listed after the equal sign.

Access to Neo4j through Python can be done two ways. The first way is to use a representational state transfer (REST) ([Fielding & Taylor, 2002](#)) web application programming interface (API) to access the data in Javascript object notation (JSON) format. The second way is to use the python-embedded driver which makes a direct straight connection. The embedded driver version is faster and allows more reads and writes and so therefore was used mostly for the backend processing. For the frontend accessing of data, the REST API was utilized.

Neo4j is ACID (Atomicity, Consistency, Isolation, Durability) compliant, meaning that all read and write operations are guaranteed. The formal definition and details of ACID can be found in the ISO standard¹.

¹ISO/IEC 9804 1998

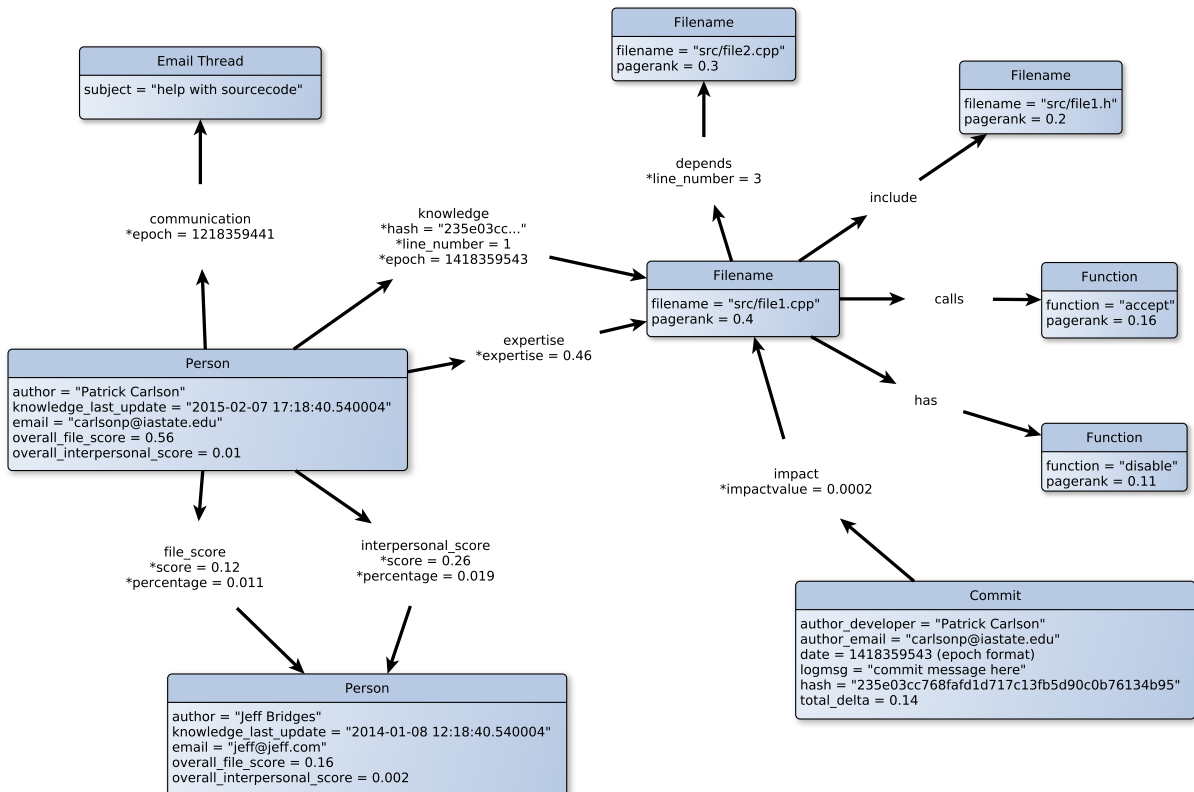


Figure 4.3 Neo4j Database Structure

- Atomicity: “A property of a set of related operations such that the operations are either all performed, or none of them are performed.”
- Consistency: “A property of a set of related operations such that the effects of the operations are performed accurately, correctly, and with validity, with respect to application semantics.”
- Isolation: “A property of a set of related operations such that partial results of the set of operations are not accessible, except by operations of the set. This definition implies that different sets of related operations that have this property and that share bound data are serializable.”
- Durability: “A property of a completed set of related operations such that all the effects of the operations are not altered by any sort of failure.”

The downside of this is that the database locks whenever an operation needs to be performed and only one connection can acquire this lock at a time. This resulted in substantial performance challenges in the development and use of the database.

The frontend user facing web portion of Jamii uses Django ([Django, 2015](#)) which is a web framework that utilizes Python. Django allows for the creation and management of webpages through a template system. Bootstrap was used to provide the theming of color and style using cascading stylesheets (CSS) ([Bootstrap, 2015](#)). Live updating of website content is done through jQuery ([jQuery, 2015](#)), Datatables ([Datatables, 2015](#)), and typeahead.js ([typeahead.js, 2015](#)). Pygments is used for source code color highlighting ([Pygments, 2015](#)). Visualization and graphics are generated using jqPlot ([jqPlot, 2015](#)) and d3.js ([D3.js, 2015](#)). User profile images are pulled from the Gravatar website ([Gravatar, 2015](#)) which ties an image to a unique email address. Some images and icons were provided by the Tango Desktop Project ([Tango Desktop Project, 2015](#)). An image of the front page of the Jamii website can be seen in [Figure 4.4](#).

4.1.1 Formative Usability Testing

The Jamii website was pilot tested with three graduate students who had not seen the website prior to the testing. Each participant was asked to explore the website and use a

Home About [Take a short survey about Jamii to help my research.](#)

Jamii

Source Code and Community Analysis


Impactful Commits

Developer Knowledge

Communication Suggestions

Find important commits relevant to you.


This section shows you code commits that have had a substantial impact to the technical dependency structure of the source code.



See Important Commits

Have a question about a file? Find an expert.


This section allows you to select a source code file to determine who has knowledge or expertise in the file. This is helpful if you have a question about a particular file or section but are unsure who to contact for help.



Find Experts

Find who you may want to email regarding your recent code contributions.

This section provides suggestions about who you (or other developers) should be increasing communication with on the mailing list based on work on shared source code.



See Suggestions

Jamii version: 0.5
 Mozilla® and Firefox® are registered trademarks of the Mozilla Foundation
 KDE® is a registered trademark of KDE e.V.

Figure 4.4 Front Page of Jamii Website

think-aloud protocol to narrate their thought process throughout the experience. After viewing the entire website, participants were interviewed about the experience. Participants offered a number of suggestions as to the usability of the website. Many of these suggestions were features or improvements that were incorporated before the website was released to the two communities, KDE and Mozilla. The details of the algorithms used in the Jamii website are outlined in the rest of this chapter.

4.2 Impactful Commits Algorithm

The impactful commits section of the website provides information about which commits have been identified as being impactful to changing the overall technical dependency structure of the source code. An example showcasing the impactful commits section for the KDE project can be seen in Figure 4.5. When the user clicks on a specific commit, the website drills down into the details of the impact calculation for that commit. This provides information as to the files and or functions and the calculated impact change for each one. An example of this can be seen in Figure 4.6. In these figures and other screenshots of the webpages in subsequent sections, developer names, profile images, and email addresses are blurred out to ensure confidentiality.

The impactful commits algorithm employs the original version of the pagerank algorithm to determine the relative importance of files and functions in the source code (Page, Brin, Motwani, & Winograd, 1998). This algorithm was utilized and popularized by the Google search engine. The concept of determining the importance of nodes in a network has multiple possible algorithms and solutions. One alternative that was examined is the calculation of an accessibility index as presented by Gould (1967). For each node in a network, a numerical value can be calculated identifying how accessible it is from all other nodes in the network (Straffin, 1980). This uses the calculation of eigenvalues and eigenvectors which is similar in structure to how pagerank can be calculated. Pagerank was selected over these other methods because the iterative calculation was easier to compute for large graphs.

A commit is a single set of changes to source code files. A commit contains the addition and deletion of the lines of source code text. A commit also has the name, email address, date, and

[Home](#)
[About](#)
[Take a short survey about Jamii to help my research.](#)

Jamii

Source Code and Community Analysis

Impactful Commits

Developer Knowledge

Communication Suggestions

Click on a commit hash to see details...

This is a list of commits that have made a substantial change to the technical dependency structure of the C++ source code.

Show the most impactful commits in the last relative to

Filter:

Date	Impact	Commit Message	Author	Hash
2014/08/10	2.38	Implemented DolphinTabWidget class to encapsulate the tab handli...	[Redacted]	dc7f2e01370899af8cab3433...
2014/02/06	2.14	Port Dolphin to Baloo Nepomuk is being replaced with Baloo	[Redacted]	5707e1e92c9c6ad320dbce5f...
2014/05/22	2.11	Keep the "free space" information updated in all visible views The ol...	[Redacted]	de197075a709057011f18b04d...
2014/07/04	1.99	Implemented DolphinTabPage class to encapsulate the split view h...	[Redacted]	6a98d83312f2b14ab878e142...
2014/06/19	0.97	Implemented DolphinRecentTabsMenu to encapsulate the recent ta...	[Redacted]	58ac6a460ec24a1e16d73c5f...
2014/07/08	0.8	Implemented (QTabBar based) DolphinTabBar class to encapsulate...	[Redacted]	7618c7943eac00836ca1e0d6...
2014/04/26	0.48	Enable the previous and next tab toolbar buttons when multiple tab...	[Redacted]	670737cbfd23fb29538af6977...
2014/07/24	0.45	UserAccount KCM: Change the name using the AccountManager In...	[Redacted]	00d66fec7aae9d50c46a6809...
2014/04/26	0.44	Port away from queryExit(), stayPreloaded() already checks whethe...	[Redacted]	0f149f4469677e8c830962c74...
2014/03/28	0.42	When you open a new tab while the search mode is enabled, the n...	[Redacted]	54208a66a07a92af968a083a...
2014/09/02	0.26	Rename "Recently Accessed" to "Recently Saved" In dolphin the Se...	[Redacted]	072c5e06cee89049ddc1553a...

Figure 4.5 Code Impact Section of Jamii Website

[Home](#)
[About](#)
[Take a short survey about Jamii to help my research.](#)

Jamii

Source Code and Community Analysis

Impactful Commits

Developer Knowledge

Communication Suggestions

Commit Impact Details

Commit Hash: dc7f2e01370899af8cab343352675ef474653
 Date: 2014/08/10
 Commit Author: [REDACTED]
 Commit Message: Implemented DolphinTabWidget class to encapsulate the tab handling from DolphinMainWindow. REVIEW: 119115

Total Impact Score: 2.376 (Substantial technical change.)

10 most impactful files/functions [\(?\)](#)

Filename	Function	Impact
kde-baseapps/dolphin/src/dolphintabpage.h	refreshViews	0.179
kde-baseapps/dolphin/src/dolphintabwidget.h	refreshViews	0.175
kde-baseapps/dolphin/src/dolphintabwidget.h	readProperties	0.175
kde-baseapps/dolphin/src/dolphintabwidget.h	saveProperties	0.175
kde-baseapps/dolphin/src/dolphintabwidget.h	openNewActivatedTab	0.175
kde-baseapps/dolphin/src/dolphintabwidget.h	openNewTab	0.175
kde-baseapps/dolphin/src/dolphintabwidget.h	currentTabPage	0.175
None	selectedItemsCount	0.173
kde-baseapps/dolphin/src/dolphintabwidget.cpp	closeTab	0.165
kde-baseapps/dolphin/src/dolphintabwidget.cpp	currentTabPage	0.165

Jamii version: 0.5
 Mozilla and Firefox are registered trademarks of the Mozilla Foundation

Figure 4.6 Code Impact Details

time of the developer who made the commit. Commits are part of the version control process and are similar in concept across version control systems and software.

Once the importance of the files and functions is determined for an individual commit, the next commit in the history is selected and the pagerank algorithm is run again. Changes to the source code could add or remove files or functions and adjust function calls. This then changes the technical structure of the overall graph resulting in different values when pagerank is run again. The delta difference between the previous page rank value and the new page rank value is calculated for each node. The sum of these delta differences becomes the final code impact value for that commit.

An example calculating pagerank for a small graph (see Figure 4.7) is shown below.

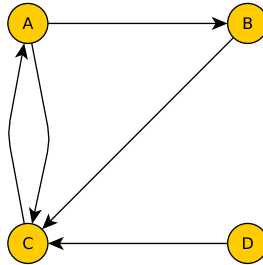


Figure 4.7 PageRank Graph Example

The variables and a definition can be seen in the following list.

- $PR(x)$: the pagerank of node x
- N : the set of nodes in the graph
- x : the node to calculate pagerank, $x \in N$
- d : a constant (usually set to 0.85), this will adjust what the minimum pagerank value can be
- W_i : the set of all nodes linking to x
- $L(i)$: the number of outgoing edges on node i , an integer value

$$PR(x) = (1 - d) + d\left(\sum_{p \in W_i} \frac{PR(p)}{L(p)}\right) \quad (4.1)$$

The calculation of pagerank values for each of the four nodes in the example graph is as follows.

$$\begin{aligned} PR(A) &= (1 - d) + d\left(\frac{PR(C)}{L(C)}\right) \\ PR(B) &= (1 - d) + d\left(\frac{PR(A)}{L(A)}\right) \\ PR(C) &= (1 - d) + d\left(\frac{PR(A)}{L(A)} + \frac{PR(B)}{L(B)} + \frac{PR(D)}{L(D)}\right) \\ PR(D) &= (1 - d) + d(0) \end{aligned}$$

Since the calculation of pagerank is iterative, the starting pagerank value is set to 1 for all nodes.

Iteration 1:

$$\begin{aligned} PR(A) &= (1 - 0.85) + 0.85\left(\frac{1}{1}\right) = 1 \\ PR(B) &= (1 - 0.85) + 0.85\left(\frac{1}{2}\right) = 0.57 \\ PR(C) &= (1 - 0.85) + 0.85\left(\frac{1}{2} + \frac{1}{1} + \frac{1}{1}\right) = 2.27 \\ PR(D) &= (1 - 0.85) + 0.85(0) = 0.15 \end{aligned}$$

Iteration 2:

$$\begin{aligned} PR(A) &= (1 - 0.85) + 0.85\left(\frac{2.27}{1}\right) = 2.08 \\ PR(B) &= (1 - 0.85) + 0.85\left(\frac{1}{2}\right) = 0.57 \\ PR(C) &= (1 - 0.85) + 0.85\left(\frac{1}{2} + \frac{0.57}{1} + \frac{0.15}{1}\right) = 1.19 \\ PR(D) &= (1 - 0.85) + 0.85(0) = 0.15 \end{aligned}$$

Iteration 3:

$$PR(A) = (1 - 0.85) + 0.85\left(\frac{1.19}{1}\right) = 1.16$$

$$PR(B) = (1 - 0.85) + 0.85\left(\frac{2.08}{2}\right) = 1.03$$

$$PR(C) = (1 - 0.85) + 0.85\left(\frac{2.08}{2} + \frac{0.57}{1} + \frac{0.15}{1}\right) = 1.64$$

$$PR(D) = (1 - 0.85) + 0.85(0) = 0.15$$

As the iterations of pagerank calculations continue, the values slowly converge. These are the final pagerank values for each node.

$$PR(A) = 1.49$$

$$PR(B) = 0.78$$

$$PR(C) = 1.58$$

$$PR(D) = 0.15$$

The same pagerank values can be seen in Figure 4.8. As can be seen, the pagerank has identified node *C* as the most important node. The impactful commits algorithm pseudocode can be seen in Algorithm 1.

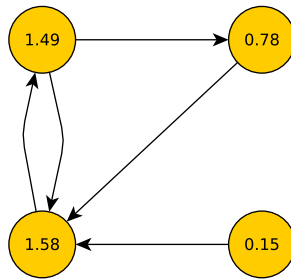


Figure 4.8 PageRank Graph Example With Values

The end result is a summed code impact value for each commit. The individual impact calculations per node are also stored in the Neo4j database. This makes it easy to show not only the summed total impact but also details as to which file or function nodes contributed the most to the impact calculation.

```

for each commit in history, starting at oldest do
  totalImpact = 0
  if calculating for first commit then
    set each node's pagerank to 1
    calculate and save pagerank for each node
  else
    calculate and update pageranks for each node
    for each node where the new pagerank  $\neq$  old pagerank do
      delta impact =  $|newPR - oldPR|$ 
      totalImpact = totalImpact + delta impact
    end for
  end if
return totalImpact
end for

```

Algorithm 1 Calculation of Code Impact

Using the original pagerank graph shown previously in Figure 4.7, the following example shows how changes to the structure of the graph impact the pagerank and the subsequent calculation of code impact. The original graph and changes to the graph via a simulated code commit can be seen in Figure 4.9. From the commit, the edge from node *C* to *A* is removed. In addition, a new node *E* is added and connected to node *D*.

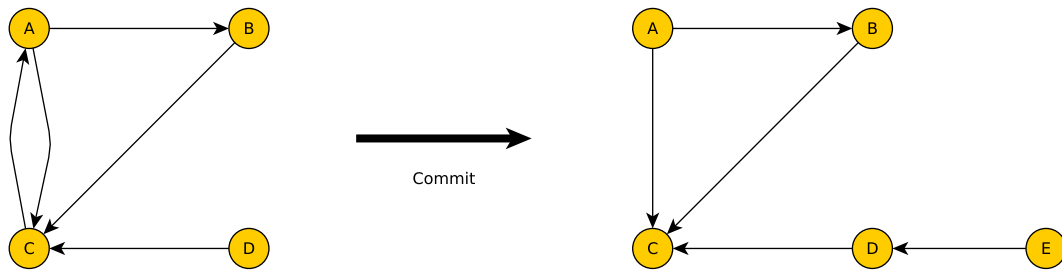


Figure 4.9 Code Impact Example

The resulting pagerank calculations for each graph are shown in Figure 4.10. As can be seen, the pagerank value for node *C* decreases dramatically because of the removed edge. This has wide ranging repercussions for not only that node but nodes that it's connected to. This is one of the main strengths of the pagerank calculation, it is able to take into account the entire structure of the graph and dependencies when calculating a value.

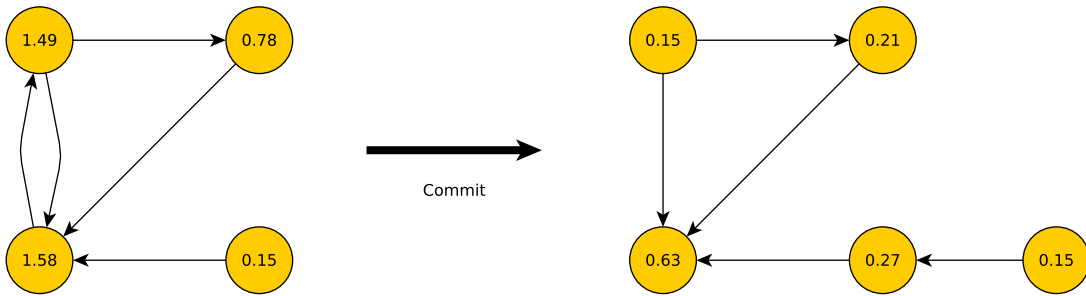


Figure 4.10 Code Impact Example Page Rank Values

Using Algorithm 1 above, we calculate the delta impact difference for each node. The sum of these delta impact values is the total code impact for this commit.

$$\mathit{delta}(A) = |1.49 - 0.15| = 1.34$$

$$\mathit{delta}(B) = |0.78 - 0.21| = 0.57$$

$$\mathit{delta}(C) = |1.58 - 0.63| = 0.95$$

$$\mathit{delta}(D) = |0.15 - 0.27| = 0.12$$

$$\mathit{delta}(E) = |0 - 0.15| = 0.15$$

$$\mathit{total\ code\ impact} = 1.34 + 0.57 + 0.95 + 0.12 = 2.98$$

By walking the history of the commits through time, the pagerank values of the new dependency graph are compared against the past pagerank values to determine what influence the commit had on the overall dependency structure.

4.3 Degree of Knowledge Algorithm

Degree of knowledge (DOK) represents the knowledge that a developer or set of developers have for a particular section of code. As developers work on a project the code changes over time. Knowledge can fluctuate based on the time spent by the developer, their technical expertise, and overall understanding of the codebase. The utility of quantifying developer expertise is especially important for new developers because it can provide a suggested contact

if there are questions about the code. Early research in knowledge recommendation systems considered knowledge as binary where only one developer has knowledge of a file. However, in reality this is a major over simplification. Multiple developers can have varying levels of experience and expertise with different sections of code in a file and this knowledge can change over time.

Previous research on identifying degree of knowledge has primarily examined changes to source code as the main factor for calculation (Mcdonald & Ackerman, 2000; Mockus & Herbsleb, 2002). One method that was developed to quantify knowledge was the “Expertise Browser” tool (Mockus & Herbsleb, 2002). The authors define, “experience atoms” (EA) which are small units of experience activity that developers contribute to a project. The experience of an object is the total combination of all experience atoms related to that object. As an example, changes in the source code could be considered experience atoms in that they apply to a specific object and are tied to the developer who initiated the change.

Mcdonald and Ackerman (2000) developed the “Expertise Recommender” tool. It uses two identification heuristics: change history and tech support. The change history heuristic is defined by the “Line 10” rule. This rule states that the person who last modified a module has the most expertise with that module. The authors contend that this developer has the “freshest” view of that module and thus would be the best person to ask when it comes to a question. The tech support heuristic identifies the importance of social interaction, communication, and identification data that is relevant in determining expertise.

More recent approaches have utilized the idea of Social Technical Congruence (Cataldo et al., 2006). The Emergent Expertise Locator (EEL) considers the importance of dependencies between files in calculating expertise (Minto & Murphy, 2007). This approach accounts for situations when a developer may use features from other files or modules but does not contribute code to them. The developer still has expertise about these modules through usage, such as in the case of an API.

Another recommendation tool is xFinder which plugs into Eclipse (Kagdi, Hammad, & Maletic, 2008). Through evaluating three pieces of information, it provides a ranked list of developers who can provide help related to a source code file. The inputs to infer developers are

commit contribution, recency of activity, and the number of active workdays. The researchers took two snapshots in time of various open source projects. The first snapshot was used for training and the second snapshot was used for validation. Through the use of these snapshots, the researchers found the accuracy of the algorithm to be between 43% and 82%.

Additional research has mined changes to the code-base that have not yet been committed (Hattori & Lanza, 2009). Syde uses a client-server architecture where local development changes are compiled and tested against the server source code. Merge conflicts can then be identified before they are pushed to the central version control. While local activity is an important facet of determining expertise, it may not be appropriate in all situations due to the need for a central server as well as potential privacy and security concerns. In large corporate environments where the platform is standardized around a particular integrated development environment (IDE), this may be feasible. However, in OSS, this may be more difficult and relying on available version control data may be more realistic.

Ownership of source code has also been tied to defects (Bird, Nagappan, Murphy, Gall, & Devanbu, 2011). In a study examining contributions to Windows Vista® and Windows 7® binary DLL files, the researchers found that sections with many minor contributors had more defects. In some cases, having a component with high ownership of a few individuals resulted in less defects as well. This study shows that the application of developer knowledge and expertise on systems can play a role in forecasting, identifying, and fixing bugs. For Open Source projects, this could mean targeting developers with high ownership to a particular area with information and having them fix bugs when they are identified. Developer expertise has also been identified through analysis of bug reports by identifying how long it takes developers to resolve bugs (Nguyen, Nguyen, Duesterwald, Klinger, & Santhanam, 2012).

While authorship is primarily used in recommendation and expertise identification systems, interaction with the source code is another important factor as highlighted by Fritz, Murphy, and Hill (2007). Through running an experiment, the researchers asked programmers questions regarding the elements that they had worked with. Using quantitative and qualitative methods, they found that the frequency and recency of interaction with the elements in the source code influenced knowledge. The same research group has also combined authorship and

interaction data in calculating degree of knowledge (Fritz, Ou, Murphy, & Murphy-Hill, 2010). Interestingly, researchers found that the most influential factor in determining DOK to be if the developer was the first author of the file. A case study and comparison to existing expertise recommenders was performed.

The idea of measuring historical changes to source code has also been applied to identifying classes for reverse engineering or refactoring. In an algorithm called, “Yesterday’s Weather”, researchers use the concept that predicting tomorrow’s weather can be influenced a fair amount by the weather pattern of today. These predictive methods were used to find classes that were good candidates as a first step for reverse engineering (Gîrba, Stéphane, & Lanza, 2004).

The line authorship expertise calculation as proposed by Gîrba, Kuhn, Seeberger, and Stéphane (2005) most closely resembles the degree of knowledge calculation proposed here. In their work, the author of each line of source code in a file was identified and the developer with the highest ratio of lines over the total lines in the file was determined to be the developer with the most expertise. The benefit of using this algorithm is that it can be calculated at multiple granularity levels. The downside is that it does not consider historical changes to the file.

Decay of expertise is something that has been examined in research throughout different areas. With regard to Social Technical Congruence, a decay factor has been proposed that reduces dependencies over time to slowly phase them out (Wagstrom et al., 2009). Other research has defined code decay as the increasing amount of effort and time necessary to make changes to the code (Eick et al., 2001; Ohlsson et al., 1999). Causes identified were inappropriate architecture, time pressure, programmer variability, and a number of other issues (Eick et al., 2001). This definition was broader in identifying causes and symptoms of code decay. In examining small group dynamics, the idea of decay has also been applied to teams showing among other things, a decline in communication as the team ages (McGrew et al., 1999).

The following proposed algorithm expands upon the work of Gîrba et al. (2005) in two ways. First, the algorithm takes into account past commits by developers. Second, the algorithm calculates knowledge based on an exponential decay of knowledge. As time passes, the developer’s memory about code they have written fluctuates. By combining this with historical

source code revision information, the algorithm calculates a model of developer knowledge. The algorithm implementation is described mathematically below.

A description of variables used in the algorithm are as follows:

- D : the set of all developers who have worked on a file.
- i : a single developer, $i \in D$.
- t : defines an integer index on the number of revisions made to a file.
- j : an integer defining a single line in a source code file.
- S_{jp} : the set of developers who have authored commits at line j from $t = 1$ to p , $p \leq t$.
- S_{jt} : the developer at line j and revision t of the source code who authored the line, $S_{jt} \in D$.
- W_t : the number of days between revision t and the current date.
- b : the exponential decay factor, ranges from $[0 - 1]$, higher values represent a higher decay amount, user specified parameter.
- x to y : a range of lines in the source code file, $0 < x \leq y$.
- O_{itxy} : the total authorship amount of developer i for a range of lines x to y at revision t in the source code file adjusted by the decay of knowledge.
- R_{ijt} : the authorship amount of developer i for a single line j in the source code file at revision t adjusted by decay of knowledge, ranges from $[0 - 1]$, see Algorithm 2.
- Z_{txy} : an integer value denoting the total potential authorship amount of all developers at revision t for a range of lines x to y in the source code.
- E_{itxy} : the percentage of knowledge for developer i at revision t for a range of lines x to y in the source code, defined as a ratio, values range from $[0 - 1]$.

As developers make commits to source code, these revisions are captured in version control. Additions, deletions, and modifications of lines of source code are applied. When a commit is made, the t value keeping track of the number of revisions increases by one.

The end goal for the algorithm is calculating developer knowledge as defined as a ratio percentage.

$$E_{itxy} = \frac{O_{itxy}}{Z_{txy}} \quad (4.2)$$

O_{itxy} has a minimum value of 0 and a maximum value of $(y - x) + 1$. Z_{txy} has a minimum value of $(y - x) + 1$ and a maximum value of $((y - x) + 1) * |D|$ where $|D|$ denotes the cardinality or number of elements in D . To determine the numerator of the knowledge ratio, the sum of the authorship amounts for a developer is calculated for each line in the source code based on the range is calculated.

$$O_{itxy} = \sum_{j=x}^y R_{ijt} \quad (4.3)$$

The knowledge algorithm presented here differs from previous work ([Girba et al., 2005](#)) by stepping back through the revisions for each line and determining the last time the author made a commit to that specific line.

```

while  $t > 0$  do
  if  $i = S_{jt}$  then
    return  $(1 - b)^{W_t}$ 
  end if
   $t = t - 1$ 
end while
return 0

```

Algorithm 2 Calculation of R_{ijt}

If the developer never authored any content for that particular line, 0 is returned. If the developer is found to have authored content for the line, the exponential decay adjusts the knowledge value that is returned. The developer's memory decays over time and this is reflected by the exponential decay calculation. This weighting value adjusts the return result

appropriately. The DOK calculation only takes into account the last commit an author made for a line. Multiple commits by the same author throughout the history of the source code line are not summed.

The denominator of the knowledge ratio is defined as the sum of the cardinality of the intersection between the list of developers and the lines of source code from the initial revision up to a certain revision point. This value increases through modification commits by developers. In this case, multiple developers now have an authorship claim over a single line in the source code.

$$Z_{txy} = \sum_{j=x}^y f(j, t) \quad (4.4)$$

$$f(j, t) = |D \cap S_{jp}|$$

An example that goes through the calculation for a small file is presented. There are three developers and knowledge will be calculated for developer A at the most recent revision 4. Overall developer knowledge will be calculated for the entire file as defined by x and y .

- $D = A, B, C$
- $i = A$
- $t = 4$
- $x = 1$
- $y = 4$
- $b = 0.01$

There were 4 revisions to the file as can be seen in Table 4.1.

The authorship information for each line in the source code file over the revisions is shown in Table 4.2. The number of days from revision t to the current date is at the top of the table at W_t .

Table 4.1 Revisions to the File

t	Operation
1	The initial commit by author A , lines 1 through 3
2	Developer B adds lines 4 and 5
3	Developer C modifies lines 1 and 2
4	One developer deletes the last line, 5

Table 4.2 Authorship Information for Source Code File

	W_t	15	10	4	2
	t	1	2	3	4
S_{jt}	$j = 1$	A		C	
	$j = 2$	A		C	
	$j = 3$	A			
	$j = 4$		B		
	$j = 5$		B		X

The deletion operation of revision $t = 4$ only has an impact on reducing the value of Z_{txy} . It has no impact which developer does the deletion. It removes that line from the knowledge calculation.

$$Z_{4,1,4} = 2 + 2 + 1 + 1 = 6 \quad (4.5)$$

At this point, the denominator of the knowledge ratio has been calculated for $E_{A,4,1,4}$. Next the numerator of the knowledge ratio needs to be calculated.

$$O_{A,4,1,4} = R_{A,1,4} + R_{A,2,4} + R_{A,3,4} + R_{A,4,4} \quad (4.6)$$

$$R_{A,1,4} = (1 - 0.01)^{15} = 0.99^{15} = 0.86$$

$$R_{A,2,4} = (1 - 0.01)^{15} = 0.99^{15} = 0.86$$

$$R_{A,3,4} = (1 - 0.01)^{15} = 0.99^{15} = 0.86$$

$$R_{A,4,4} = 0$$

Then the values for each line are summed to yield the final value.

$$O_{A,4,1,4} = 0.86 + 0.86 + 0.86 + 0 = 2.58 \quad (4.7)$$

The ratio is finalized and the knowledge for developer A is complete.

$$E_{A,4,1,4} = \frac{O_{A,4,1,4}}{Z_{4,1,4}} = \frac{2.58}{6} = 0.43 \quad (4.8)$$

In this example, developer A has a calculated knowledge of approximately 43%. Note that because of the exponential decay, the knowledge sum of all developers will not usually be 100%. As time progresses and developers leave the code untouched, the knowledge calculation needs to be recomputed. This allows the user to compare across the files to see relative to others what developers have been working on. This example shows the past history of authorship on the part of developer A in determining knowledge. The influence of the increasing number of days since the authorship reduces the knowledge through exponential decay.

This algorithm is integrated into the Jamii website in the developer knowledge section. The initial screen allows the user to navigate the files and folders of the source code as seen in Figure 4.11. Once the user selects a file, they are presented with another page that allows them to select the lines in the file that they wish to calculate knowledge for. This can be seen in Figure 4.12.

4.4 Communication Suggestions Algorithm

The “communication suggestions” algorithm provides suggestions about who a developer could be increasing communication with on the mailing list based on shared technical dependencies and recent communication. Using the basic tenet of STC, the algorithm tries to identify gaps in congruence between the current communication structure and the ownership or knowledge between developers on files.

Home About [Take a short survey about Jamii to help my research.](#)

Jamii

Source Code and Community Analysis

Impactful Commits Developer Knowledge Communication Suggestions

Select a source code file... (.lua, .cpp, .h, .xml, .py, .txt, .mkd, .sh, .js, .c, .hpp, .hh, .inl, .in, ...)

kde-baseapps /

- doc
- dolphin
- kdepasswd
- kdialog
- keditbookmarks
- kfind
- konq-plugins
- konqueror
- lib
- nsplugins
- plasma
- CMakeLists.txt
- COPYING
- COPYING.DOC
- COPYING.LIB
- CTestConfig.cmake
- ConfigureChecks.cmake
- Mainpage.dox
- README
- config-apps.h.cmake

Figure 4.11 Developer Knowledge Section of Jamii Website

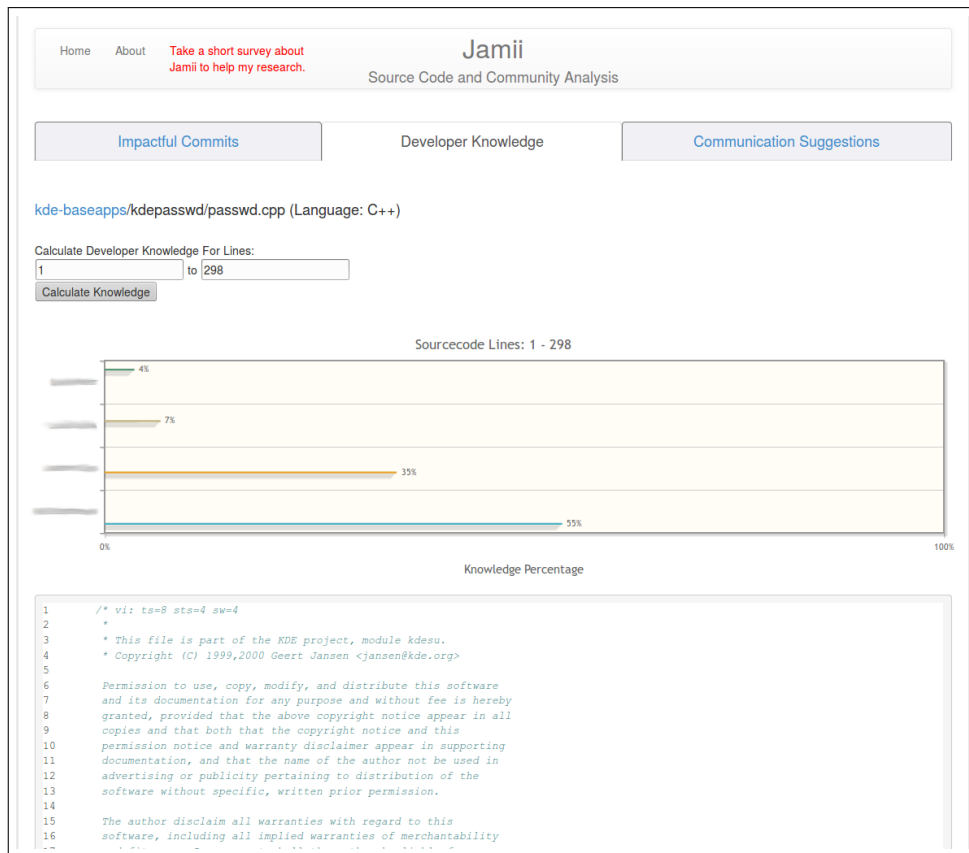


Figure 4.12 Developer Knowledge Details

On the Jamii website, the user first selects their name, or the name of someone of whom they would like to view communication suggestions. This can be seen in Figure 4.13. The next page provides the details and the top five people that it has deemed most important for increasing communication on the email mailing list. An example of this can be seen in Figure 4.14.

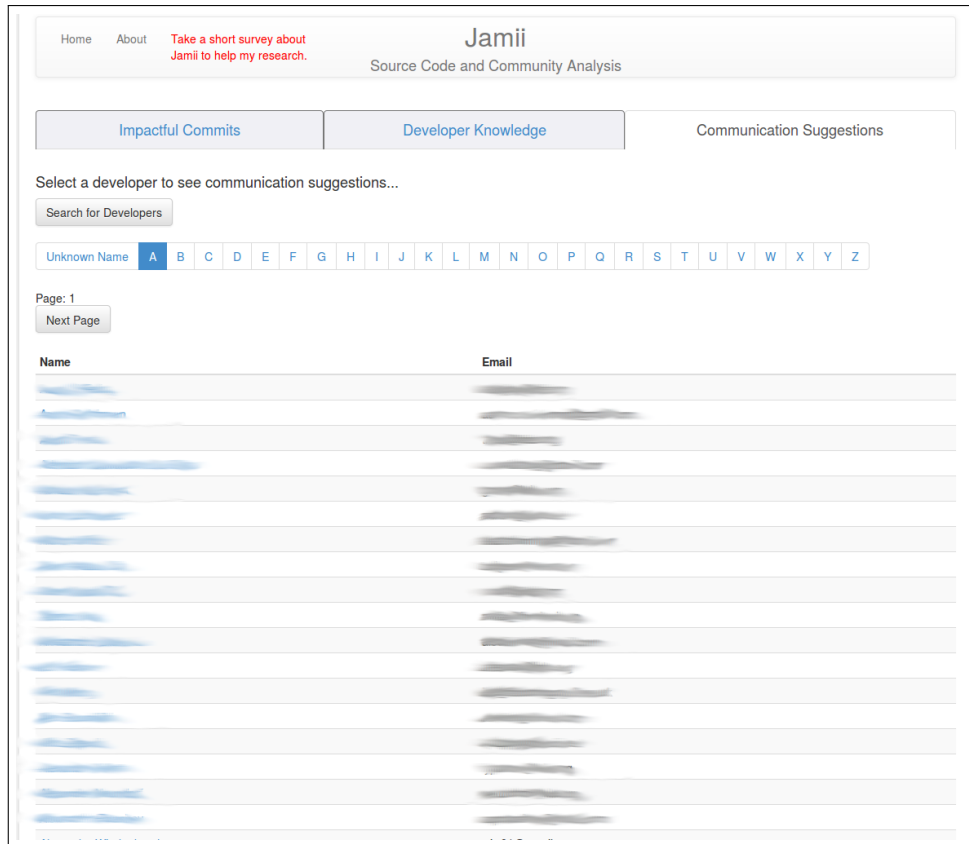


Figure 4.13 Communication Suggestions of Jamii Website

First, the backend processing identifies the email communication that has occurred via various email threads on the mailing lists. Each developer or author may have contributed one or more emails to the thread. In addition, the knowledge each developer has for each line of source code is available from the previous knowledge calculation algorithm. An example can be seen in Figure 4.15.

The next step is to break these multiple links down to a single value between author \rightarrow email thread and author \rightarrow file. The same knowledge algorithm that was described earlier in this chapter is used to calculate the knowledge of the author for the entire file.

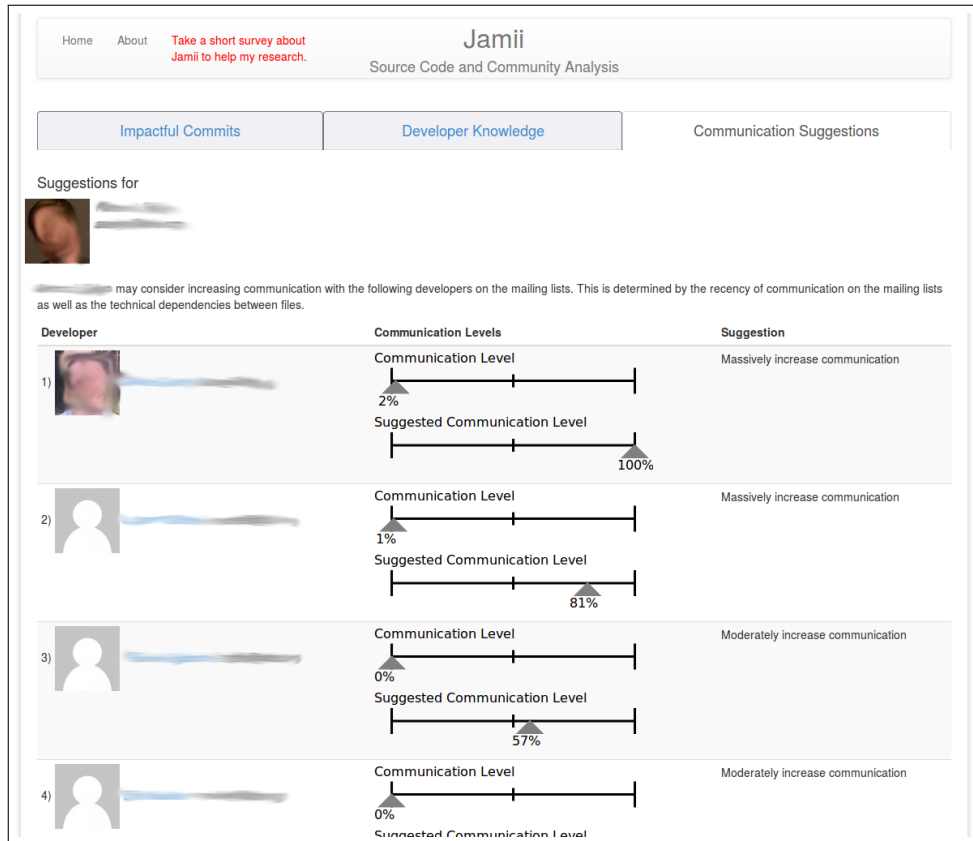


Figure 4.14 Communication Suggestions Details

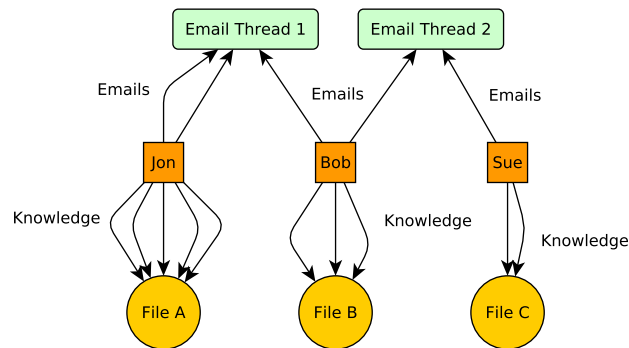


Figure 4.15 Communication Suggestions Initial Data

- b : the exponential decay factor, ranges from $[0 - 1]$, set to 0.01
- x : the first line in the file, set to 1
- y : the last line in the file, must be ≥ 1

$$O_{itxy} = \sum_{j=x}^y R_{ijt}$$

Using the same method of exponential decay, a communication score is generated between an author and each email thread where they have been a contributor. This singular score accounts for the recency of the email(s) as well as the number that are sent. It uses the same concepts as seen in the knowledge calculation. The final calculated communication score is defined as C_{iq} and can be seen in Algorithm 3. This is done for all developers and the corresponding email communications.

- E_q : the number of days between the email thread q and the current date
- b : the exponential decay factor, ranges from $[0 - 1]$, set to 0.01
- i : a single developer, $i \in D$
- q : a single email thread
- C_{iq} : the final communication score from developer i to email thread q

```

 $C_{iq} = 0$ 
for each communication edge from developer  $i$  to email thread  $q$  do
     $C_{iq} = C_{iq} + (1 - b)^{E_q}$ 
end for
return  $C_{iq}$ 

```

Algorithm 3 Calculation of C_{iq}

At this point, the knowledge and communication edges have been consolidated to single scores as can be seen in Figure 4.16.

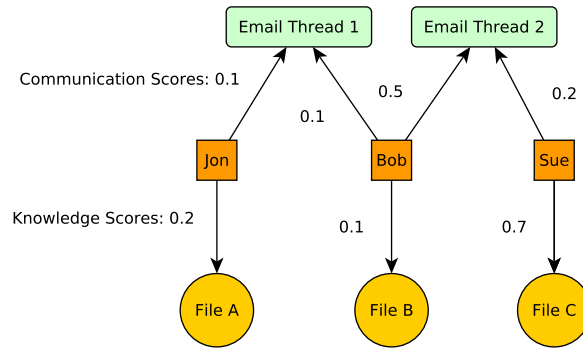


Figure 4.16 Consolidation of Knowledge and Communication Scores

The next step is to calculate two different summed scores. The interpersonal score represents the amount of directed communication between one person and another. The file score represents the amount of shared technical dependencies between two developers. The usage of these two scores is similar in concept to the idea of calculating congruence. Two developers with a high file score show that they should be communicating a fair amount about their work. If the interpersonal score between these two developers is low, this represents low communication and thus low congruence.

The interpersonal score is calculated by summing the communication scores that run between the authors and can be seen in Algorithm 4.

```

for each developer  $v$  do
  for each developer  $p$  do
     $interpersonalscore_{v,p} = 0$ 
    for each email communication exchange between
       $v$  and  $p$  via an email thread do
       $interpersonalscore_{v,p} = interpersonalscore_{v,p} + communicationscore_v$ 
    end for
    return  $interpersonalscore_{v,p}$ 
  end for
end for

```

Algorithm 4 Calculation of interpersonal score for each developer

In the database, the interpersonal score is a directed edge between two developers. The interpersonal scores between $v \rightarrow p$ versus $p \rightarrow v$ may be, and often are, different.

The file score is calculated by summing the knowledge values from one author to another modified by a weighting factor and can be seen in Algorithm 5. The weighting factor varies depending on whether two developers are working on the same file, files that are dependent on one another, or files that share a common dependent file. The weighting means that longer chains of files linking two authors will be modified lower in score because there is less of a dependency relationship between those two people. An example of the three possible conditions on which the file score is used and modified by the weighting can be seen in Figure 4.17.

```

for each developer  $v$  in the database do
  for each developer  $p$  in the database do
     $filescore_{v,p} = 0$ 
    for each knowledge link between  $v$  and  $p$  via a file(s) do
      if there is direct knowledge between  $v$  and  $p$  via a file then
         $filescore_{v,p} = filescore_{v,p} + knowledge_v$ 
      else if there is an knowledge connection between
         $v$  and  $p$  through dependent files then
         $filescore_{v,p} = filescore_{v,p} + (knowledge_v * 0.5)$ 
      else if there is an knowledge connection between
         $v$  and  $p$  through three dependent files then
         $filescore_{v,p} = filescore_{v,p} + (knowledge_v * 0.25)$ 
      end if
    end for
  return  $filescore_{v,p}$ 
end for
end for

```

Algorithm 5 Calculation of file score for each developer

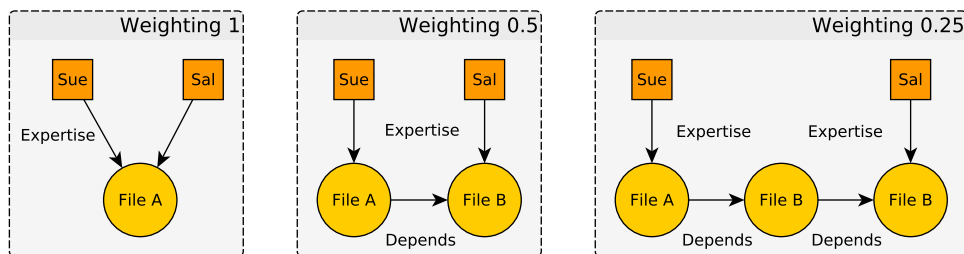


Figure 4.17 Weighting File Scores

In the database, the file score is a directed edge between two developers. The file scores between $v \rightarrow p$ versus $p \rightarrow v$ may be (and often are) different.

At this point, the interpersonal scores and file scores have been tabulated for all the developers. The scores are then normalized as can be seen in Algorithm 6. Since the interpersonal scores and file scores are calculated differently and are based on different data, comparing the raw values cannot be done. Normalizing the scores ensures that comparisons between the two can be made.

```

for each developer  $v$  do
   $IPmin$  = minimum interpersonal score for  $v$ 
   $IPmax$  = maximum interpersonal score for  $v$ 
   $FSmin$  = minimum file score for  $v$ 
   $FSmax$  = maximum file score for  $v$ 
  for each interpersonal score  $g$  of developer  $v$  do
     $normalizedinterpersonalscore = \frac{g-IPmin}{IPmax-IPmin}$ 
  end for
  for each file score  $h$  of developer  $v$  do
     $normalizedfilescore = \frac{h-FSmin}{FSmax-FSmin}$ 
  end for
end for

```

Algorithm 6 Calculation of normalized interpersonal and file scores

The resulting normalized interpersonal and file scores have a range from $[0 - 1]$. In the database, the normalized interpersonal and file scores are directed edges between two developers. The normalized interpersonal and file scores between $v \rightarrow p$ versus $p \rightarrow v$ may be, and often are, different.

By normalizing the scores, they can now be compared against one another. The normalized interpersonal score from one developer to another represents how much that developer has been communicating with the other developer relative to their total communication. The normalized file score from one developer to another represents how much that developer works on similar files as the other developer relative to other developers.

After the scores are normalized, communication suggestions can be determined. Considering the main tenet of STC is that the congruence between the technical dependencies and the

communication between developers should be high, direct comparisons can be made between the two normalized scores. As an example, assume the normalized interpersonal score for developer v to p is 0.2. This means that of all the communication that developer v has done, there has been overall very little communication to developer p . Next assume the normalized file score for developer v to p is 0.9. This indicates a high amount of shared programming through files and file dependencies between the two developers. The Jamii website shows the difference between these two normalized scores and displays the top five largest differences where the file score is larger than the interpersonal score. This shows where increased communication could help developers by increasing their interpersonal score.

CHAPTER 5. WEBSITE ANALYSIS

Overview

The Jamii website was tested with data from two large open source communities, Mozilla and KDE. The processing and calculations took a series of days for both communities. Substantial work and testing was done to speed up the calculation process. Some of this effort involved running some of the processing on faster department computers while other efforts involved parallelizing portions of the calculation process.

After the data was processed and the Jamii website went online, the survey and interviewing process started. The objective of this data collection was to assess the perceived usefulness of the website by community members as well as the potential for improving the socialization process for new members. Overall survey and interview data collection started on March 30, 2015 and ended April 28, 2015 resulting in time period of around one month.

5.1 Jamii Data

The Mozilla version control data was pulled from the gecko-dev repository on Github¹ covering the master branch from December 10, 2014² to January 9, 2015³. The gecko-dev repository contains various applications and libraries such as the web-browser Firefox, Netscape Portable Runtime (NPR), extensions, xulrunner, and many others. The email mailing list data was pulled from archives from the Gmane website ([Gmane, 2015](#)). Limitations of processing time were encountered because the knowledge calculations need to run on every line of a file going back through the source code history. Therefore, the developer knowledge processing

¹<https://github.com/mozilla/gecko-dev.git>

²Git Hash: 437abd21eec991a1fd8d0ac8ee85bde4749e3a71

³Git Hash: c4485f2b8f28eae3ee50c82d6f35cad125478cae

was limited to the “browser” source code folder. Developer knowledge was also limited to files with the following file extension (.cpp, .h, .c, .hpp, .hh, .inl, .in).

- gmane.comp.mozilla.devel.builds
- gmane.comp.mozilla.devel.firefox
- gmane.comp.mozilla.devel.gaia
- gmane.comp.mozilla.devel.mobile
- gmane.comp.mozilla.devel.platform
- gmane.comp.mozilla.firefox.devel

The KDE version control data was pulled from the KDE baseapps repository⁴ covering the master branch from Jan. 1, 2014⁵ to Jan. 5, 2015⁶. The kde-baseapps repository contains various applications such as the file manager Dolphin, web-browser Konqueror, utility tools such as KFind, and others. The email mailing list data was pulled from archives from the Gmane website (Gmane, 2015). Developer knowledge was limited to files with the following file extension (.lua, .cpp, .h, .xml, .py, .txt, .mkd, .sh, .js, .c, .hpp, .hh, .inl, .in).

- gmane.comp.kde.devel.core
- gmane.comp.kde.devel.core.bugs
- gmane.comp.kde.devel.general
- gmane.comp.kde.general

The resulting finalized Neo4j databases for Mozilla and KDE ended up having almost 60,000 nodes each and hundreds of thousands of edges. The details of which can be seen in Table 5.1. The resulting database size for KDE is slightly misleading as being small compared to the overall size of the community because of how the source code repositories are organized. Mozilla for

⁴git://anongit.kde.org/kde-baseapps

⁵Git Hash: ce040ab74f812cca04ed6f7dae02e8a13cfcfc04

⁶Git Hash: c78c6104bfcaf507a5a0915eec12e4ba05fc3ba0

example has the source code for most of their projects in a single repository called “gecko-dev”. This “gecko-dev” repository was the one analyzed and presented through the Jamii website. KDE on the other hand has decided to split up their source code into many different repositories. This analysis was run on the “kde-baseapps” repository but there are others such as “kdelibs”, “kdegames”, “kdenetwork” and a wide variety of others ([KDE Getting Started, 2015](#)).

Table 5.1 Neo4j Database Statistics

Nodes/Edges	Mozilla	KDE	Total
Mailing List Threads	6710	45729	52439
Functions	26626	850	27476
Files	20913	895	21808
People	2483	12346	14829
Commit Hashes	826	109	935
Total Nodes	57559	59930	117489
Expertise	1199	5417	6616
Knowledge Lines	81624	232311	313935
Depends	81846	1296	83142
Has	25511	813	26324
Calls	30556	908	31464
Include	60301	1296	61597
Communication	27156	203686	230842
FileScore	234572	70796	305368
InterpersonalScore	51223	286096	337319
Impact	50735	1962	52697
Total Edges	644723	804581	1449304

Analysis of the two source code repositories using the Count Lines of Code (CLOC) program version 1.60 ([CLOC, 2015](#)) was performed. The results for Mozilla can be seen in Table 5.2 and the results for KDE can be seen in Table 5.3. These results were used to check against the Neo4j database statistics to ensure a rough match in terms of the number of files.

The Mozilla codebase has a large number of C/C++ files, but also a large number of Javascript files. One downside is that the technical dependencies were not identified for files besides C++. Adding additional dependency checks for other programming languages is one

Table 5.2 CLOC Mozilla Statistics

Language	Files	Blank	Comment	Code
C++	7819	560777	448802	3038699
Javascript	19853	406474	386086	2273592
C	3053	246768	345311	1426339
HTML	31452	164787	37507	1424489
C/C++ Header	10894	316974	620420	1406972
Python	1783	62228	74484	241480
Java	1991	48222	88496	226315
Bourne Shell	377	26239	30624	172131
Assembly	364	24277	18377	136513
JavaServer Faces	2323	10190	3111	98017
IDL	1208	13272	0	95107
XML	1783	10828	5661	88929
CSS	826	17555	6766	76242
m4	91	5143	894	45094
Objective C++	169	9981	7491	43257
NAnt scripts	729	4864	0	30032
Perl	98	3276	5004	17130
YAML	11	1421	116	11719
DTD	197	2210	2927	7617
make	128	2193	2524	7258
Teamcenter def	39	35	12	4781
SKILL	4	68	2	2419
DOS Batch	30	322	109	1973
yacc	3	203	164	1956
Objective C	16	420	523	1864
MATLAB	14	348	346	1825
XSD	3	143	1129	1587
XSLT	19	130	115	1108
CMake	7	208	498	953
lex	4	209	74	937
Pascal	5	245	500	775
Korn Shell	5	83	165	526
Bourne Again Shell	18	118	211	511
Expect	6	105	164	506
Ant	2	27	109	401
Lisp	2	40	37	228
SASS	3	39	0	217
awk	2	41	8	154
PowerShell	2	29	110	110
Ruby	3	22	6	83
sed	6	16	25	63
C Shell	2	13	9	32
Ada	1	5	0	16
CoffeeScript	2	7	4	13
D	2	7	40	10
PHP	1	0	0	2
Total	85350	1940562	2088961	10889982

area for future improvement on the processing side of Jamii. CLOC also identified a wide number of other supporting file types in a range of programming languages.

Table 5.3 CLOC KDE Statistics

Language	Files	Blank	Comment	Code
C++	379	20246	13951	91218
C/C++ Header	369	7639	15131	18996
CMake	92	712	219	1505
XML	16	13	14	1112
Python	2	66	174	943
C	1	116	50	605
HTML	8	63	120	449
Bourne Shell	29	24	5	166
Ruby	2	0	12	114
CSS	2	24	12	60
Total	900	28903	29688	115168

The kde-baseapps codebase is mostly comprised of C/C++ files which made it ideal for identifying the technical dependencies. There are a few other supporting scripts and file-types.

5.1.1 Website Analytics

Website analytics were calculated using Awstats ([Awstats, 2015](#)). Awstats is a program that parses webserver access logs to identify when and what files were served to visitors. There were over 250 visits to the Jamii website over approximately a one month time period. Four different values were calculated by Awstats, the definitions of which are seen below ([Awstats Documentation, 2015](#)). The number of unique visitors, visits, pages, and hits can be seen in Table 5.4. The number of visits over the one month time period can be seen graphically in Figure 5.1. The graph shows high usage the first few days that gradually tapers off. These aggregate statistics are the combined results for both Mozilla and KDE.

- **Unique Visitors:** This is a unique person or host that has made at least one hit on the website. The uniqueness is determined by the IP address of the visitor.
- **Visits:** A visit can be thought of as analogous to a session. A unique visitor could visit the website and then come back later in the day. This would be counted as two separate visits. One visit usually results in multiple pages and hits.

- **Pages:** A page is an HTML or overall aggregate viewable page rendered to the visitor. Images or other types of supporting files such as Javascript, CSS, and other files are not included in this calculation.
- **Hits:** Hits are any files requested from the server. This includes files that are counted as pages. Images and any other supporting files such as Javascript, CSS, or others are included in this calculation.

Table 5.4 Total Analytics Results

Unique Visitors	Visits	Pages	Hits
225	258	1779	4753

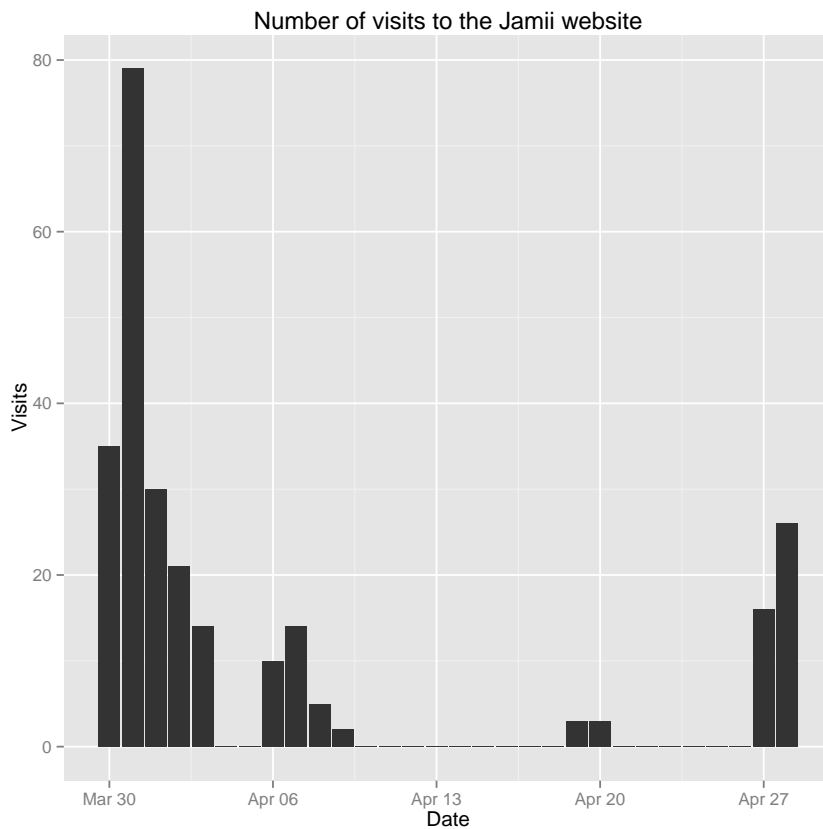


Figure 5.1 Number of Visits to the Jamii Website

5.2 Quantitative Survey Results

Solicitation messages to participate in the survey and/or participate in an interview were sent out through community mailing lists and IRC. In addition, posts were made in subreddits on the popular social media and news aggregation website Reddit ([Reddit, 2015](#)).

The development of the survey was focused mainly on identifying how the three algorithms and subsequent sections on the website could help them in improving their understanding of the community and codebase. In addition, multiple questions were created to identify the impact Jamii could have on the socialization process. These Likert responses were planned to be combined into a single factor. An additional question asked participants to rate their experience level in the community. Participants would then be grouped into beginners and veterans, similar to what was done in the initial survey. An analysis could then be performed to see if there were perceptual differences between beginners and veterans on these factors.

The survey sent out to participants can be seen in Appendix B. Unfortunately, the response rate on the surveys for both communities was low, hampering the ability to run analysis. After around two weeks of gathering quantitative and qualitative results, the decision was made to focus primarily on performing interviews and gathering qualitative results. The additional detail and depth of the survey responses was felt to be the most helpful in assessing the usefulness of the tool and the potential impact on socialization.

5.3 Qualitative Interview Results

A semi-structured survey was developed to gather more specific data as to the perceived usefulness of the Jamii website by community members in Mozilla and KDE. A semi-structured interview meant participants were mostly asked questions from a set list, however, the interviewer was able to ask additional questions based on the responses of the participant. All interviews were done using text-based chat on IRC and took approximately twenty minutes. Solicitation to participate in the interview was done through community mailing lists, IRC, gathering email addresses from the survey results, and directly emailing members of community engagement efforts. The set of predefined questions are listed below.

- “Can you go over your involvement with {KDE, Mozilla}? How did it get started?”
- “Would you discuss how you joined the community. Were there any challenges in joining? If so, what were they?”
- “How long did you use Jamii and over what period of time?”
- “Was there anything you liked about Jamii? If so, what?”
- “Was there anything you disliked about Jamii? If so, what?”
- “Do you think Jamii could improve the socialization process for new developers? If so, in what ways?”
- “Is there anything you would add or change in Jamii to improve the socialization process for new developers? If so, in what ways?”
- “Jamii has three different sections: ‘impactful commits’, ‘developer knowledge’, and ‘communication suggestions’. Can you discuss your thoughts on each of these three sections.”
- “Of the three sections listed in the previous question, which one is most helpful or would be most helpful for new members wishing to join {KDE, Mozilla} and why?”
- “Has Jamii changed your perception of your involvement in the community? Or the involvement of others?”

The development of these questions was done with consultation with other researchers to try to ensure a minimal level of bias in the questions themselves. The questions were meant to be fairly open ended in order to enable the interviewee the flexibility to answer as they saw fit. Interview bias covers the interviewer who may ask biased or leading question and the interviewee who may respond in ways that they feel the interviewer expects as opposed to their true feelings.

Five interviews were collected over the one month period. Three people from KDE and two people from Mozilla were interviewed. The interviewee’s name, username, and any identifying comments have been removed in order to protect confidentiality. Interviewees from KDE are

designated as K1, K2, and K3. Interviewees from Mozilla are designated as M1 and M2. A selection of questions, associated responses, and discussion of the responses are listed below. Some corrections were made to the spelling and grammar of responses to ease readability. Not all questions were asked of every participant because of time constraints.

- *Can you go over your involvement with {KDE, Mozilla}? How did it get started?*
 - K1: [Worked on various KDE projects. Participated in a Google Summer of Code (GSOC) project.]
 - K2: [Started hanging out in IRC and then worked on various KDE applications.]
 - K3: [C/C++ developer. Works on various KDE applications.]
 - M1: [Worked on development of a Firefox extension before being hired by Mozilla.]
 - M2: [Works in a non-software development role for Mozilla.]

Details and direct quotes from interviewees are not presented here in order to protect confidentiality. However, many of the respondents work on the project in some sort of development fashion. All interviewees had at least one year in their respective community and many had eight or more years of participation.

- *Were there any challenges in joining? If so, what were they?*
 - K1: “Not really, it was as smooth as possible. I respected a lot the teams and they were very encouraging and open.”
 - K3: “Yes and no: I am a person that teaches himself the stuff he needs to do or want to do - in this case it was the development for KDE. This was more or less quite easy. The more complex and difficult topics were these areas that weren’t documented, for example release management and translation stuff. For many people, these workflows are normal, so there is no need to document them, but for me this was the most difficult stuff. But all in all it was just a matter of asking people/have discussions with them.”

These comments reflect similar results from the initial survey on KDE socialization. The KDE community socialization is smooth and people are knowledgeable. The comment by K3 about improving documentation of workflows and processes especially for non-traditional software development areas such as release management and translation is an area that KDE may want to consider examining.

- *Was there anything you liked about Jamii? If so, what?*
 - K1: “The ‘impactful commits’ section can be curious to figure out what the project is about, I’m unsure about the metrics it used though, so it’s hard to tell if it makes sense.”
 - K2: “The web design is quite modern and it’s easy to use.”
 - K3: “Yes. I use git quite heavily. In git there is a command called ‘git blame’ - with this command you can see who worked on a specific piece/lines of code (History management). Jamii provides something similar - but with a nice graphical view.”
 - M1: “I liked the developer knowledge section, although when I tried it, I couldn’t find any files that it actually showed information about. But from the video [a short introductory video on the main page] (of the kde version), it looked useful.”
 - M2: “I really like the impact points and the way you can filter bugs according to impact. I’m not sure how that is calculated but I think it is really cool to see how much of an impact your work had.”

The impactful commits section was a feature that many interviewees mentioned as being helpful or potentially helpful. Comments by K3 as well as comments by an individual on IRC who did not participate in the interview process mentioned the similarity to “git blame”. There are certainly many similarities to this command and the information that is provided. The intent was that the website would be faster and more precise with the ability to see the code and select the appropriate section than having to manually do this through “git blame”. M2 appears to be talking about the “impactful commits” section, however, incorrectly refers to the commits as bugs.

- *Was there anything you disliked about Jamii? If so, what?*
 - K1: “Developer knowledge is quite bare. It’s already a common thing to git blame/log a file and checked who worked on what it could be interesting to get a project-global outlook, but this is not what this one is about.”
 - K2: “A couple of things, I tried to look at developer knowledge, which worked fine for source files, but complained that the .docbook files are binary, though they are plain xml. [Interviewee provides link to a specific Jamii page.] The button at the top here says ”Unknown Name” though the name is known, it’s just not an ascii letter so probably should say ‘Non-Ascii letters’ or ‘Unknown Letter’.”
 - K3: “Not liked - no. I would call it more ‘not perfect’. As a person with a technical background, the tab ‘Impactful commits’ was quite interesting. Sadly I saw no reason why a specific commit was marked as important (aka why did the algorithm mark it as important - important dev? large patch? etc.). reason = information”
Interviewer: OK, so if the algorithm was clarified, do you think it would be a useful feature?
“Yes. Personally I could better classify/rank the patch for myself. Sometimes there are really long (several weeks) discussions about a patch that changes 4 lines - so how should someone handle this? If Jamii gives more information, it would be more clear.”
 - M1: “Maybe it’s just me as a not-really-development person, but the communication suggestions section doesn’t seem all that useful to me. It also shows pretty much the same information for every person I’ve checked. It also doesn’t really say *why* one should communicate more with these people.”
 - M2: “Sorry I was just looking around, but there’s nothing that I dislike in particular.”

Similar to the previous question responses, K1 mentioned “git blame” as an alternative. K2 had trouble analyzing developer knowledge for certain types of files. The reason all files were not analyzed was it would have taken longer from a computational time standpoint. Therefore,

primarily file extensions associated with C/C++ were selected for analysis. Multiple comments by interviewees mentioned that they wanted additional details as to how the calculation was done and/or how the analysis was coming up with these results. During the development of the Jamii website, a fair amount of thought was put into deciding how much information and what information should be presented via the website. Providing large amounts of details would have resulted in a deluge of information and cluttered the clean design of the website. However, having more details and information might have increased the trust level of developers. The decision was made to go with a very minimal presentation and try to abstract away the details of the calculations, however, based on the interview responses, this will probably be something that will be changed in future versions of the website.

- *Do you think Jamii could improve the socialization process for new developers? If so, in what ways?*

- K1: “‘communication suggestions’ again is quite obscure, also probably disregards quite some mailing lists, so it’s hard to take seriously the results. I don’t think it could, as it is right now. Community analysis is a good tool but it needs lots of love. We’ve tried to use Bitergia stuff in the past (which is a bit more advanced) and even there we couldn’t use it like we would have wanted because we need to make sure all of the information is actually in there.”

- K2: “Also, I’m not sure what the use of pages like this are, after clicking on a commit hash. [Interviewee provides a link to the impactful commits details of a specific commit on the Jamii website.] It shows filenames and function names, but has no links to the actual sources on cgit or such. It could help new developers see who works on code, but the same could really be seen in git blame also to be honest, or git log or whatnot.”

Interviewer: Do you think software developers in particular want to know more of the details and analysis behind the calculations?

“Yeah, definitely.”

- K3: “I am not sure if I am the right person to answer this question. I can just speak for myself: Developers and many person in a techy-environment are ‘lazy’ people - means they like the fastest and most efficient way. Most of the times, websites are outranked by command line programs like git, cli review board. Also we don’t like to change our workflow. So to give you an answer: I don’t think so. Not because your project is bad or missing functionality. It’s more because we live in our ‘now’ ecosystem. Maybe if you can provide it as command line application or something it would be really interesting.”
- M1: “If it could tie in to bugzilla’s ‘good first bugs’ classification to find good bugs for new developers to work on, and then from there figure out which files need to be touched to fix the bug, and from there find relevant developers for them to talk to about any problems they encounter, that’d be helpful, I think.”
- M2: “Yes I do. I really like the part about finding who are the experts. I think that really facilitates communication. Also communication suggestions are pretty cool. Although something like a way to contact directly the people who are listed in your suggested communication would be nice. I’m not sure if that’s the intent of the website though.”

K1 references Bitergia, a company that is doing data analysis on open source projects and creating online dashboards ([Bitergia, 2015](#)). K1 and other interviewees via other questions mention some of the challenges of using these types of tools to analyze data. Ensuring all of the information has been parsed and analyzed can be difficult from a wide variety of technical standpoints. Making decisions or conclusions based on incomplete data analysis could result in inaccurate adjustments by community members or the community as a whole. One question then arises, is the use of incomplete or partial data to provide suggestions at least better than having no suggestions in the first place?

K3 provides some interesting insight into the possible thought processes of developers and technically savvy individuals. In the comment, K3 says that developers want to take the most

efficient route and usually this is through direct access of the information. This seems to be a very telling comment and may explain some of the low turnout by developers to use Jamii.

- *Is there anything you would add or change in Jamii to improve the socialization process for new developers? If so, in what ways?*
 - K1: “I’m unsure about that one, having metrics is mostly important for the community managers, to see what parts of the project are aching more than newcomers, but then I’m missing something.”
 - K2: “Making it integrated with cgit, i.e. make hash urls go to the git web interface of the respective project would be useful in my opinion. And also possibly tying it’s name lookup to any upstream name services, like the names could be links to the user’s accounts on identiy.kde.org or similar for other projects.”
 - K3: “I would try to embed it into an existing system/eco system, so people can use it ‘on the fly’. But this is really my own impression/idea and takes much time.”
 - M1: “Maybe Jamii could expose sections of code that are frequently worked on so that new developers can know that there’s lots of people with the ability to help them out?”
 - M2: “Definitely something like add a button next to one’s profile through which you can ping them on IRC or email them. And that can be recorded and calculated towards your communication level. Otherwise people need to make an extra effort to socialize outside the tool and that is sometimes hard.”

Many of the interviewees mentioned integration with existing websites and tools as part of the ecosystem of the respective projects. While this certainly would help, the programming and design of the Jamii website and processing side was meant to be generic for any project that can provide source code and email data. One option would be to refactor the code to allow plugins and customization from the base analysis. This would allow projects to tailor the analysis and display to their respective ecosystems while keeping the base analysis project agnostic.

- *Jamii has three different sections: “impactful commits”, “developer knowledge”, and “communication suggestions”. Can you discuss your thoughts on each of these three sections.*
 - K2: “The ‘communication suggestions’ is a bit confusing, it seems to just list anyone who ever committed to the project. Impactful commits uses some metric to determine impact, but doesn’t say what that metric is measuring that I could tell. Otherwise it seems to work. Developer knowledge seems to be just an aggregation of git blame. With the name ‘developer knowledge’ I was expecting it to contain api documentation or something.”
 - M1: “The Impactful Commits section should probably provide links to the actual commits in the repository so users of Jamii can see what actually changed. Can’t comment too much on Developer Knowledge as I’ve not found many files that it actually exposed any information. Communication suggestions seems to show the same five people no matter who I select, and doesn’t really give any information about why we should talk with these people.”

As mentioned previously, more details as to how the calculations are performed could be helpful. The preference between providing a minimal interface to the data analysis and providing all the details relevant to the calculation is an area that could use additional scrutiny through testing.

- *Of the three sections: “impactful commits”, “developer knowledge”, and “communication suggestions”, which one is most helpful or would be most helpful for new members wishing to join {KDE, Mozilla} and why?*
 - K1: “Probably the impactful commits, to have an overview of where things are going.”
 - K2: “Probably impactful commits, since it helps find the person most involved with the project.”

- K3: “I have to give you an answer that contains two ‘answers’: 1.) The ‘impactful commits’ tab, because new person can see what drives/controls a community (What are the hot topics etc.) 2.) The communication suggestion, because it serves as ‘git blame’. If you are working on a piece of code and need information about the author of the code, this is really useful.”
- M1: “Developer Knowledge probably exposes the most useful information for new contributors. New contributors are likely going to be coming from bugzilla where there’s usually information provided (at least in the ‘good first bug’ case) about what files need to be touched to fix the bug. If the new contributor could then look up those files in Jamii’s Developer Knowledge section to see who knows about those file, they could then take that information and reach out to those developers with any questions they have.”
- M2: “Developer knowledge because you can find experts in certain fields that can help show you around. It is very important to have some point of contacts if you’re new.”

K3 mentions “communication suggestions” as being tied to “git blame” but it’s assumed this is in reference to developer knowledge. Overall, the verdict for the section that would be most helpful for new developers is fairly split between the impactful commits and developer knowledge sections.

The overall message seems to be that the website was helpful and the interviewees felt the features were useful. The general participation rates for both the survey and interviews were lower than expected. Therefore, some of the focus on questions was on improvements that could be made. After looking through the responses, there appear to be two major important points.

The first point is that many of the interviewees, especially those in developer roles want more details as to how the calculations were done. The challenge of presenting all the information and overloading users versus having a very clean and minimal presentation was something that was debated during the development of Jamii. More work could be done to display additional

detail of the meaning behind the calculation while at the same time trying not to burden the user with content to read. By providing these additional details, the trust in the results should hopefully rise.

The second point that may have influenced low trust or adoption of the Jamii website is developer workflow and direct access to information. As mentioned directly by one of the interviewees and indirectly by others via their explanation of the use of “git blame”, developers have existing tools and workflows in place for finding information. Developers may feel that their current direct access method is the most efficient and may be less willing to try alternative methods. It’s possible that community members, in particular software developers, who have direct access to the source code and a variety of version control tools to manage, it may trust methods that they know work and are reliable as opposed to looking at a website where the methods are not as transparent. The hesitation or perhaps reluctance to use these types of aggregation websites and dashboards to present information comes across in these interviews as well as through comments by people on IRC.

One of the challenges in the future development and deployment of source code and community analysis tools for software development is balancing the presentation of the detail involved in the calculation of results with simplicity and ease-of-use of the website. These improvements, and others will need to be studied in order to increase adoption of these types of websites/tools by open source communities and developers in general. There is still tremendous future potential in the analysis of the wide variety of data sources created as part of the software development process to improve understanding.

CHAPTER 6. SUMMARY AND DISCUSSION

A variety of research contributions have been developed as part of this work. Mozilla and KDE, two large open source communities were surveyed and analyzed. While the two communities seem to be doing a reasonable job of managing the community and ensuring new users and developers get the attention they need, there are always areas for improvement.

The results suggest community members spend around a quarter to a third of their time communicating. Less experienced members of the community seem to have a more positive outlook than veteran members on factors such as member respect, the socialization process, and some measures of Transactive memory. This is important for ensuring new developers have realistic expectations which might otherwise make them leave. Qualitative comments from survey participants indicated a wide variety of possibilities for improving the socialization process as well as expanding or improving available tools. Practically all of the comments by community members for how they joined the community were positive for KDE. However, for Mozilla, the socialization process seemed to be more difficult.

Both open source communities have large numbers of developers, a large codebase, and an extensive amount of communication occurs. This can make the socialization process harder because there are potentially additional hurdles on the technical and social fronts. However, this also makes these communities ripe for the use and adoption of tools to better understand the technical and social structure.

As part of this effort, three algorithms were created and an online website called Jamii was developed that displays the results from the algorithms. The website helps developers track important recent commits. These “impactful commits” are identified based on the technical dependency structure of the C++ files and their relative importance in the entire network. The website provides the ability to identify developers with a high amount of knowledge for

a given file or even a small section of a file. This means developers have a starting point for who to contact if they have a question. And lastly developers can identify potential gaps in their communication based on commits to shared or dependent source code files. Ensuring that developers keep the right people up-to-date with changes that they make will increase the overall congruence of the community and result in a healthier project.

Interviews and the resulting qualitative data showed that in general participants felt the Jamii website was helpful. While overall adoption and usage of the website was fairly low, two factors were identified through the interviews that may help improve the website in the future. The first is that developers wanted more detail and background on how the calculations and analysis were performed. Presenting sufficient information for developers through the website to ensure they trust and understand the algorithm results while at the same time not overloading users with details is challenging and may require more development iterations. The second factor revolves around developer workflow and their knowledge of existing tools. Some developers may be unwilling to use new tools and websites to help them if they feel that current tools are more efficient. Ensuring that these types of analytical dashboard websites are helpful not only at a high level for community managers but also for the average developer will be important.

6.1 Algorithm Limitations and Assumptions

As with any algorithm, there are some limitations and assumptions with each of the three that were developed. For the “impactful commits” algorithm, the technical structure was only defined through includes and calls for C++ files. This means other types of dependencies from other programming languages were not included. This is not so much a problem of the algorithm as it is a challenge in the implementation. The algorithm needs the technical structure as input and it is up to the backend processing to provide this. Support for additional programming languages could be provided in the future.

The developer knowledge algorithm uses the history of individual lines of source code as a measurement of knowledge. It’s possible for a commit to change a large number of source code lines by automated means. For example, a commit could change the line endings or the

indentation of files and would result in a high calculation of knowledge for that individual but in reality, this change was purely automatic. This would result in an incorrect calculation of knowledge.

The knowledge algorithm also does not understand the substance of the change. This would require a much more nuanced understanding of the commit. For example, if a commit only changes one line, the calculation of knowledge will be only adjusted for that line resulting in a minimal adjustment of overall knowledge. However, the one change may have been linked to a deep understanding of the entire file and dependent technical files if it was a bug fix. Understanding the magnitude of a change could provide a more detailed picture as to developer knowledge.

The communication suggestions algorithm assumes two developers with a high degree of shared technical files should be communicating a great deal. The algorithm does not know the contents of the emails and is only considering emails in the analysis. Live IRC chat or other methods of communication are not utilized. Because of this, these suggestions could potentially be inaccurate. Adding additional sources and methods of communication could increase this accuracy.

6.2 Future Work

In addition to programming and presentation improvements to the website, there are a wide variety of improvements and research that could be done at a higher level to answer important research questions. Algorithm improvements could include changing the decay amount b to vary based on the developer. For example, developers who have recently committed changes or who have looked at the code recently could have less exponential decay applied. In the algorithm, a developer who commits multiple changes to the same line will only be counted for the most recent change. This could be altered to apply the calculation for all edits the developer made through the life of the file. Structural changes of the source code are also not considered. For instance, identifying if there is a modification on an interface or if a conditional is adjusted would influence the gravity of the change and its effect on dependent systems. This

would be a particularly challenging item to address because of the difficulty of identifying the change.

Another area of improvement is the display of data already captured and processed by the existing three algorithms. The file scores and interpersonal scores that are calculated as part of the communication suggestions algorithm could be presented and visualized to developers. This could give an indication of what source code files a developer works on at a high level. In addition, it could provide an indication of who the developer talks to.

One more area for future work is to add in data on bug fixes. This could be correlated with existing processed data to see if developer knowledge can be used as a prediction for where bugs might be occurring. Source code commits that have a large technical structure impact could also be more likely to result in bugs. Flagging these or other areas for additional review could be helpful in reducing the number of bugs in the software and ensuring stability and security of the product.

The concept of congruence itself could also be calculated and examined over time. Since the communication and technical structure are both always changing, the resulting high level aggregated congruence value for the entire community as well as individual levels are constantly changing. By showing developers how these values change, they can see the impact of their own efforts to improve congruence. In addition, prediction could be done to attempt to predict future congruence values. By doing so, this prediction could provide a glimpse into the future state of the software and community.

One other possibility is to turn the Jamii website and analysis into a service that is provided not only to software developers but anyone doing large scale collaborative projects. The company or organization would provide the data and all the analysis work would be done through the service. This could form the basis for future performance metrics to help managers have a better understanding of the collaboration that is occurring as well as what areas might need improvement. These and other items could be worked on in the future to provide additional information through the Jamii website and to better understand the development process.

6.3 Conclusion

While there are some limitations and assumptions that went into the creation of these algorithms, they are still a good starting point for educating developers about the core importance of STC and the relationship between work on source code files, technical dependencies, and communication. The results indicated that participants felt the displayed information and knowledge was helpful but that the presentation through the website could use some iterations and improvement. The design of the algorithms was done such that they can be applied not just to open source software development, but to any collaborative effort. As long as there are people working together on artifacts that have dependencies, the overall core idea of congruence still holds.

The aim of the Jamii website was not to be targeted at a high level to community managers, but directly to the developers themselves. This research aims to provide not aggregated statistics of the number of commits or emails sent but to give developers actionable information. These algorithms increase developer understanding by showing them what's important as well as help them recognize their importance in the overall community. Large scale data analysis like this is most helpful when it can provide small and direct personalized suggestions that are targeted to the needs of the individual.

This research will hopefully serve as a starting point for the adoption of increasingly powerful online dashboards and analysis tools not only for open source projects where access to data is easy, but other collaborative efforts. The development of these algorithms and testing provide contributions and broader implications that are valuable to the STC community of researchers, the open source community, and comprehensive efforts to understand group collaboration.

APPENDIX A. COMMUNITY SURVEY

Mozilla Community Survey

Introduction

The information obtained through this survey will be used to model the demographics, technical profile, and culture of individuals associated with the Mozilla community. Any information you provide will be used anonymously and referenced using only an identification number during publication. If you do not wish to answer a question, simply leave it blank. The email address(es) you provide will be used to link this survey data to publicly available data regarding Mozilla such as mailing list posts, bug reports, and source code commits. In addition, the email address will be used to communicate possible future surveys or results.

You must be 18 years of age or older to participate in this survey.


What is your age (years)? You must be 18 years or older to participate in this survey.

What is the email address that you use for Mozilla? Please enter the email address or addresses that you use for the mailing list, bug reports, and/or source code commits. If you use multiple email addresses, please list all of them. We will NOT distribute this email address to any other group.

Spambot check...

already *stapler*

Type the two words

 [Privacy & Terms](#)

Demographic

What is your gender identity?

- Male
- Female
- Other identification
- Prefer not to specify

Where do you live? / Country of residence

How many spoken/written languages are you proficient in?



List the top three spoken/written languages that you are most proficient in.

Language #1

Language #2

Language #3

Indicate the highest degree or level of school/education that you have completed. If currently enrolled, mark the previous entry.

- No schooling
- Elementary School / Primary School
- High School / Secondary School
- Associate Degree
- Professional Degree
- Undergraduate / Bachelor's Degree / Post-Secondary
- Master's Degree
- Doctorate Degree

OSS Experience

How long have you been a member of the Mozilla community? Make sure to indicate whether your response is in days, weeks, months, or years.

How often do you use Free/Libre Open Source software?

- Never Rarely Sometimes Often Always
-

How much experience do you have contributing in Free/Libre Open Source communities?

- None A Little Some Quite a Bit A Lot
-

How much computer programming experience do you have in comparison to the average Open Source contributor?

- None Less Than Average An Average Amount More Than Average A Lot
-

How many Free/Libre Open Source communities are you currently involved in?



How many minutes per week do you spend on the following activities as part of the Mozilla community?

Programming New Features	<input type="text"/>
Programming to Fix Bugs	<input type="text"/>
Filing New Bug Reports	<input type="text"/>
Responding to Existing Bug Reports	<input type="text"/>
Reading Code Written By Others	<input type="text"/>
Writing Documentation	<input type="text"/>
Communication (email, IRC, etc.)	<input type="text"/>
Other	<input type="text"/>

Do you receive significant monetary compensation to contribute to Mozilla? (For example as part of your employment.)

Yes
 No

How integrated are you into the Mozilla community?

An Outsider	A Beginner	Somewhat Integrated	Reasonably Integrated	An Expert Member	A Veteran Member	A Core Member
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Where would you like to end up in the Mozilla community?

An Outsider	A Beginner	Somewhat Integrated	Reasonably Integrated	An Expert Member	A Veteran Member	A Core Member
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Have you posted to one or more of the Mozilla mailing lists?

Yes
 No

How long did you watch the mailing list and read email posts before your first email post? Make sure to indicate whether your response is in days, weeks, months, or years.

Barriers

How easy is it for a new members to join the Mozilla community?

Very Difficult Difficult Somewhat Difficult Neutral Somewhat Easy Easy Very Easy

How would you rate the following skills/characteristics of a new member wishing to join the Mozilla community?

	Not at all Important	Very Unimportant	Somewhat Unimportant	Neither Important nor Unimportant	Somewhat Important	Very Important	Extremely Important
Technical Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Programming Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Social Communication Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Management	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

New members are quickly and easily integrated into the Mozilla community.

Strongly Disagree Disagree Somewhat Disagree Neither Agree nor Disagree Somewhat Agree Agree Strongly Agree

The Mozilla community treats new members with respect.

Strongly Disagree Disagree Somewhat Disagree Neither Agree nor Disagree Somewhat Agree Agree Strongly Agree

How well are you able to effectively track the following Mozilla processes?

	Very Ineffective	Ineffective	Somewhat Ineffective	Neither Effective nor Ineffective	Somewhat Effective	Effective	Very Effective
Mailing Lists	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bug Reports	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Source Code Changes	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Events	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Releases	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Milestones	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Transactive Memory

Each community member has specialized knowledge of some aspect of the project.

Strongly Disagree Disagree Neither Agree nor Disagree Agree Strongly Agree

I know which community members have expertise in specific areas.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

I trust that other community members' knowledge about the project is credible.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

I am confident relying on the information that other community members bring to the discussion.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

The community accomplishes tasks smoothly and efficiently.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

There is a lot of confusion about how the community accomplishes tasks.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Comments

Are there ways that Mozilla can support new/potential members in their desire to join?

What tools or functionality (new or existing) would help you contribute to Mozilla?

What were your experiences in joining Mozilla?

Please enter any comments, questions, or suggestions below.

KDE Community Survey

Introduction

The information obtained through this survey will be used to model the demographics, technical profile, and culture of individuals associated with the KDE community. Any information you provide will be used anonymously and referenced using only an identification number during publication. If you do not wish to answer a question, simply leave it blank. The email address(es) you provide will be used to link this survey data to publicly available data regarding KDE such as mailing list posts, bug reports, and source code commits. In addition, the email address will be used to communicate possible future surveys or results.

You must be 18 years of age or older to participate in this survey.

What is your age (years)? You must be 18 years or older to participate in this survey.

What is the email address that you use for KDE? Please enter the email address or addresses that you use for the mailing list, bug reports, and/or source code commits. If you use multiple email addresses, please list all of them. We will NOT distribute this email address to any other group.

Spambot check...

Type the two words

Demographic

What is your gender identity?

- Male
- Female
- Other identification
- Prefer not to specify

Where do you live? / Country of residence

How many spoken/written languages are you proficient in?



List the top three spoken/written languages that you are most proficient in.

Language #1

Language #2

Language #3

Indicate the highest degree or level of school/education that you have completed. If currently enrolled, mark the previous entry.

- No schooling
- Elementary School / Primary School
- High School / Secondary School
- Associate Degree
- Professional Degree
- Undergraduate / Bachelor's Degree / Post-Secondary
- Master's Degree
- Doctorate Degree

OSS Experience

How long have you been a member of the KDE community? Make sure to indicate whether your response is in days, weeks, months, or years.

How often do you use Free/Libre Open Source software?

- Never Rarely Sometimes Often Always
-

How much experience do you have contributing in Free/Libre Open Source communities?

- None A Little Some Quite a Bit A Lot
-

How much computer programming experience do you have in comparison to the average Open Source contributor?

- None Less Than Average An Average Amount More Than Average A Lot
-

How many Free/Libre Open Source communities are you currently involved in?



How many minutes per week do you spend on the following activities as part of the KDE community?

Programming New Features	<input type="text"/>
Programming to Fix Bugs	<input type="text"/>
Filing New Bug Reports	<input type="text"/>
Responding to Existing Bug Reports	<input type="text"/>
Reading Code Written By Others	<input type="text"/>
Writing Documentation	<input type="text"/>
Communication (email, IRC, etc.)	<input type="text"/>
Other	<input type="text"/>

Do you receive significant monetary compensation to contribute to KDE? (For example as part of your employment.)

Yes
 No

How integrated are you into the KDE community?

An Outsider	A Beginner	Somewhat Integrated	Reasonably Integrated	An Expert Member	A Veteran Member	A Core Member
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Where would you like to end up in the KDE community?

An Outsider	A Beginner	Somewhat Integrated	Reasonably Integrated	An Expert Member	A Veteran Member	A Core Member
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Have you posted to one or more of the KDE mailing lists?

Yes
 No

How long did you watch the mailing list and read email posts before your first email post? Make sure to indicate whether your response is in days, weeks, months, or years.

Barriers

How easy is it for a new members to join the KDE community?

Very Difficult Difficult Somewhat Difficult Neutral Somewhat Easy Easy Very Easy

How would you rate the following skills/characteristics of a new member wishing to join the KDE community?

	Not at all Important	Very Unimportant	Somewhat Unimportant	Neither Important nor Unimportant	Somewhat Important	Very Important	Extremely Important
Technical Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Programming Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Social Communication Skills	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Management	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

New members are quickly and easily integrated into the KDE community.

Strongly Disagree Disagree Somewhat Disagree Neither Agree nor Disagree Somewhat Agree Agree Strongly Agree

The KDE community treats new members with respect.

Strongly Disagree Disagree Somewhat Disagree Neither Agree nor Disagree Somewhat Agree Agree Strongly Agree

How well are you able to effectively track the following KDE processes?

	Very Ineffective	Ineffective	Somewhat Ineffective	Neither Effective nor Ineffective	Somewhat Effective	Effective	Very Effective
Mailing Lists	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bug Reports	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Source Code Changes	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Events	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Releases	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Milestones	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Transactive Memory

Each community member has specialized knowledge of some aspect of the project.

Strongly Disagree Disagree Neither Agree nor Disagree Agree Strongly Agree

I know which community members have expertise in specific areas.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

I trust that other community members' knowledge about the project is credible.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

I am confident relying on the information that other community members bring to the discussion.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

The community accomplishes tasks smoothly and efficiently.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

There is a lot of confusion about how the community accomplishes tasks.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Comments

Are there ways that KDE can support new/potential members in their desire to join?

What tools or functionality (new or existing) would help you contribute to KDE?

What were your experiences in joining KDE?

Please enter any comments, questions, or suggestions below.

APPENDIX B. JAMII SURVEY

Mozilla: Jamii Survey

Default Question Block

Thank you for taking the time to complete this short survey about your experience with the Jamii website. It should take less than 5 minutes and any information you provide will be used anonymously. If you do not wish to answer a question, simply leave it blank. You must be 18 years of age or older to participate.

Please read the following [informed consent document](#) before proceeding.

Have you spent 5 minutes or more using the Jamii website?

- Yes
 No

OK. Please try out the Jamii website first and then come back to take the survey. Thank you.

Are you 18 years of age or older?

- Yes
 No

Indicate your experience level within the Mozilla community, from novice to expert.

Novice | | Expert

Use this reminder image of the Jamii website to answer the questions below.

[Home](#) [About](#) Take a short survey about Jamii to help my research.

Jamii

Source Code and Community Analysis

Impactful Commits

Developer Knowledge

Communication Suggestions

Click on a commit hash to see details...

This is a list of commits that have made a substantial change to the technical dependency structure of the C++ source code.

Show the most impactful commits in the last relative to

Filter:

Date	Impact	Commit Message	Author	Hash
2014/08/10	2.38	Implemented DolphinTabWidget class to encapsulate the tab handl...	[redacted]	dc7f2e01370899af8cab3433...
2014/02/06	2.14	Port Dolphin to Baloo Nepomuk is being replaced with Baloo	[redacted]	5707e1e92c9c6ad320d8e5f...
2014/05/22	2.11	Keep the "free space" information updated in all visible views The of...	[redacted]	de197075a70905701118b04d...
2014/07/04	1.99	Implemented DolphinTabPage class to encapsulate the split view h...	[redacted]	6a96d83312f2b14ab878e142...
2014/06/19	0.97	Implemented DolphinRecentTabsMenu to encapsulate the recent ta...	[redacted]	58ac5a460ec24a1e16c73c5f...
2014/07/08	0.8	Implemented (QTabBar based) DolphinTabBar class to encapsulate...	[redacted]	7618c7943eac00836ca1e0d5...
2014/04/26	0.48	Enable the previous and next tab toolbar buttons when multiple tab...	[redacted]	670737cfd23fb29538af6977...
2014/07/24	0.45	UserAccount KCM: Change the name using the AccountManager In...	[redacted]	00966fec7aae9d50c46a6809...
2014/04/26	0.44	Port away from queryExit(), slayPreloaded() already checks whethe...	[redacted]	01149f4489677e9c830962c74...
2014/03/28	0.42	When you open a new tab while the search mode is enabled, the n...	[redacted]	54206a66a07a92a966a083a...
2014/09/02	0.25	Rename "Recently Accessed" to "Recently Saved" in dolphin the Se...	[redacted]	072c5e06cee89049ddc1553a...

The "impactful commits" section on the Jamii website would help me understand significant changes to the Mozilla codebase.

Strongly Disagree | | Strongly Agree

The "impactful commits" section on the Jamii website would help me understand significant changes to the Mozilla codebase **for areas of the code that I work on.**

Strongly Disagree | | Strongly Agree

The "impactful commits" section on the Jamii website would help me understand the work of other programmers working in my area.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "impactful commits" questions below. e.g. What led you to answer as you did?

Use this reminder image of the Jamii website to answer the questions below.

Home About **Take a short survey about Jamii to help my research.**

Jamii

Source Code and Community Analysis

Impactful Commits Developer Knowledge Communication Suggestions

[kde-baseapps/kdialog/widgets.cpp](#) (Language: C++)

Calculate Developer Knowledge For Lines:
 to

Sourcecode Lines: 1 - 402

Knowledge Percentage
3%
97%

```

1 //
2 // Copyright (C) 1990 Matthias Hoeizer <hoeizer@kde.org>
3 // Copyright (C) 2002-2005 David Faure <faure@kde.org>
4 //
5 // This program is free software; you can redistribute it and/or modify
6 // it under the terms of the GNU General Public License as published by
7 // the Free Software Foundation; either version 2 of the License, or
8 // (at your option) any later version.
9 //
10 // This program is distributed in the hope that it will be useful,
11 // but WITHOUT ANY WARRANTY; without even the implied warranty of
12 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 // GNU General Public License for more details.
14 //
15 // You should have received a copy of the GNU General Public License
16 // along with this program; if not, write to the Free Software
17 // Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

```

The "developer knowledge" section on the Jamii website would help improve my ability to find who's familiar with a specific source code file.

Strongly Disagree | | Strongly Agree

The "developer knowledge" section on the Jamii website would increase my confidence in reaching out to members of the developer community for help.

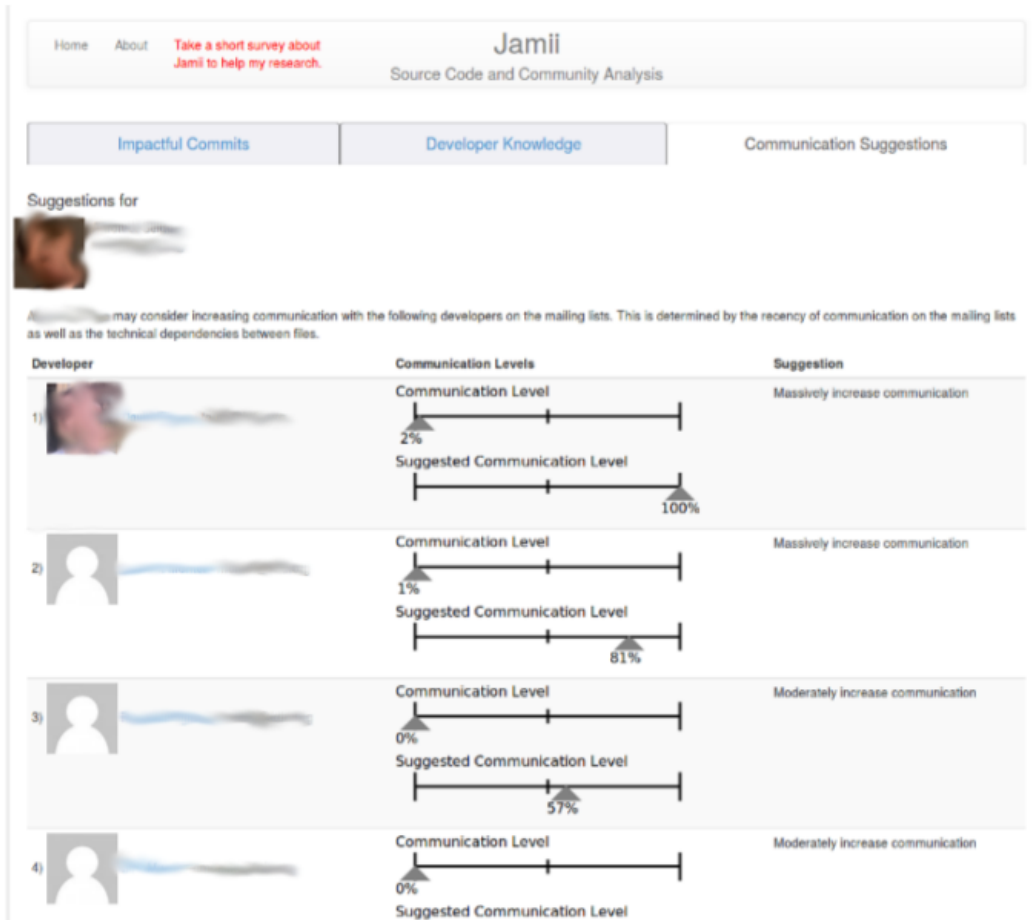
Strongly Disagree | | Strongly Agree

The "developer knowledge" section on the Jamii website would change the process I use to seek help and ask questions in the developer community.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "developer knowledge" questions below. e.g. What led you to answer as you did?

Use this reminder image of the Jamii website to answer the questions below.



The "communication suggestions" section on the Jamii website would increase my confidence in communicating with the appropriate people on the mailing list.

Strongly Disagree | | Strongly Agree

The "communication suggestions" section on the Jamii website would change the process I use to communicate on the mailing list.

Strongly Disagree | | Strongly Agree

The "communication suggestions" section on the Jamii website would change the process I use to communicate using other methods such as IRC.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "communication suggestions" questions below. e.g. What led you to answer as you did?

The Jamii website in general would increase my integration and involvement in the community.

Strongly Disagree | | Strongly Agree

The Jamii website in general would increase my understanding of the overall codebase.

Strongly Disagree | | Strongly Agree

The Jamii website in general would result in an increase in the number of new developers in the community.

Strongly Disagree | | Strongly Agree

The Jamii website in general would increase the retention of existing developers in the community.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above questions below. e.g. What led you to answer as you did?

What is the email address that you use for Mozilla? The email address(es) you provide (optional) will be used to link this survey data to publicly available data of Mozilla such as mailing list posts, bug reports, and source code commits. If you use multiple email addresses, please list all of them. We will NOT distribute this email address to any other group.

I'm conducting one-on-one interviews via instant chat with people who have used Jamii. This is completely optional. Would you be willing to have me email you to setup a time for an interview?

- Yes
 No

If you have any questions, comments, or concerns please list them below. Thank you again and feel free to contact me via email for any questions or details: carlsonp@iastate.edu

KDE: Jamii Survey

Default Question Block

Thank you for taking the time to complete this short survey about your experience with the Jamii website. It should take less than 5 minutes and any information you provide will be used anonymously. If you do not wish to answer a question, simply leave it blank. You must be 18 years of age or older to participate.

Please read the following [informed consent document](#) before proceeding.

Have you spent 5 minutes or more using the Jamii website?

- Yes
 No

OK. Please try out the Jamii website first and then come back to take the survey. Thank you.

Are you 18 years of age or older?

- Yes
 No

Indicate your experience level within the KDE community, from novice to expert.

Novice | | Expert

Use this reminder image of the Jamii website to answer the questions below.

[Home](#) [About](#) [Take a short survey about Jamii to help my research.](#)

Jamii

Source Code and Community Analysis

Impactful Commits

Developer Knowledge

Communication Suggestions

Click on a commit hash to see details...

This is a list of commits that have made a substantial change to the technical dependency structure of the C++ source code.

Show the most impactful commits in the last relative to

Filter:

Date	Impact	Commit Message	Author	Hash
2014/08/10	2.38	Implemented DolphinTabWidget class to encapsulate the tab handl...	[blurred]	dc7f2e01370899af8cab3433...
2014/02/06	2.14	Port Dolphin to Baloo Nepomuk is being replaced with Baloo	[blurred]	5707e1e92c9c6ad320dcae5f...
2014/05/22	2.11	Keep the "free space" information updated in all visible views The of...	[blurred]	de197075a70905701118b04d...
2014/07/04	1.99	Implemented DolphinTabPage class to encapsulate the split view h...	[blurred]	6a96d83312f2b14ab878e142...
2014/06/19	0.97	Implemented DolphinRecentTabsMenu to encapsulate the recent ta...	[blurred]	58ac5a460ec24a1e16c73c5f...
2014/07/08	0.8	Implemented (QTabBar based) DolphinTabBar class to encapsulate...	[blurred]	7618c7943eac00836ca1e0d5...
2014/04/26	0.48	Enable the previous and next tab toolbar buttons when multiple tab...	[blurred]	670737cfd23fb29538af6977...
2014/07/24	0.45	UserAccount KCM: Change the name using the AccountManager In...	[blurred]	00966fec7aae9d50c46a6809...
2014/04/26	0.44	Port away from queryExit(), slayPreloaded() already checks whethe...	[blurred]	01149f4489677e9c830962c74...
2014/03/28	0.42	When you open a new tab while the search mode is enabled, the n...	[blurred]	54206a66a07a92a966a083a...
2014/09/02	0.25	Rename "Recently Accessed" to "Recently Saved" in dolphin the Se...	[blurred]	072c5e06cee89049ddc1553a...

The "impactful commits" section on the Jamii website would help me understand significant changes to the KDE codebase.

Strongly Disagree | | Strongly Agree

The "impactful commits" section on the Jamii website would help me understand significant changes to the KDE codebase **for areas of the code that I work on.**

Strongly Disagree | | Strongly Agree

The "impactful commits" section on the Jamii website would help me understand the work of other programmers working in my area.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "impactful commits" questions below. e.g. What led you to answer as you did?

Use this reminder image of the Jamii website to answer the questions below.

Home About **Take a short survey about Jamii to help my research.**

Jamii

Source Code and Community Analysis

Impactful Commits
Developer Knowledge
Communication Suggestions

[kde-baseapps/kdialog/widgets.cpp](#) (Language: C++)

Calculate Developer Knowledge For Lines: to

Sourcecode Lines: 1 - 402

Category	Percentage
Sourcecode Lines: 1 - 402	97%
Other	3%

Knowledge Percentage

```

1 //
2 // Copyright (C) 1990 Matthias Hoeizer <hoeizer@kde.org>
3 // Copyright (C) 2002-2005 David Faure <faure@kde.org>
4 //
5 // This program is free software; you can redistribute it and/or modify
6 // it under the terms of the GNU General Public License as published by
7 // the Free Software Foundation; either version 2 of the License, or
8 // (at your option) any later version.
9 //
10 // This program is distributed in the hope that it will be useful,
11 // but WITHOUT ANY WARRANTY; without even the implied warranty of
12 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 // GNU General Public License for more details.
14 //
15 // You should have received a copy of the GNU General Public License
16 // along with this program; if not, write to the Free Software
17 // Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
                
```

The "developer knowledge" section on the Jamii website would help improve my ability to find who's familiar with a specific source code file.

Strongly Disagree | | Strongly Agree

The "developer knowledge" section on the Jamii website would increase my confidence in reaching out to members of the developer community for help.

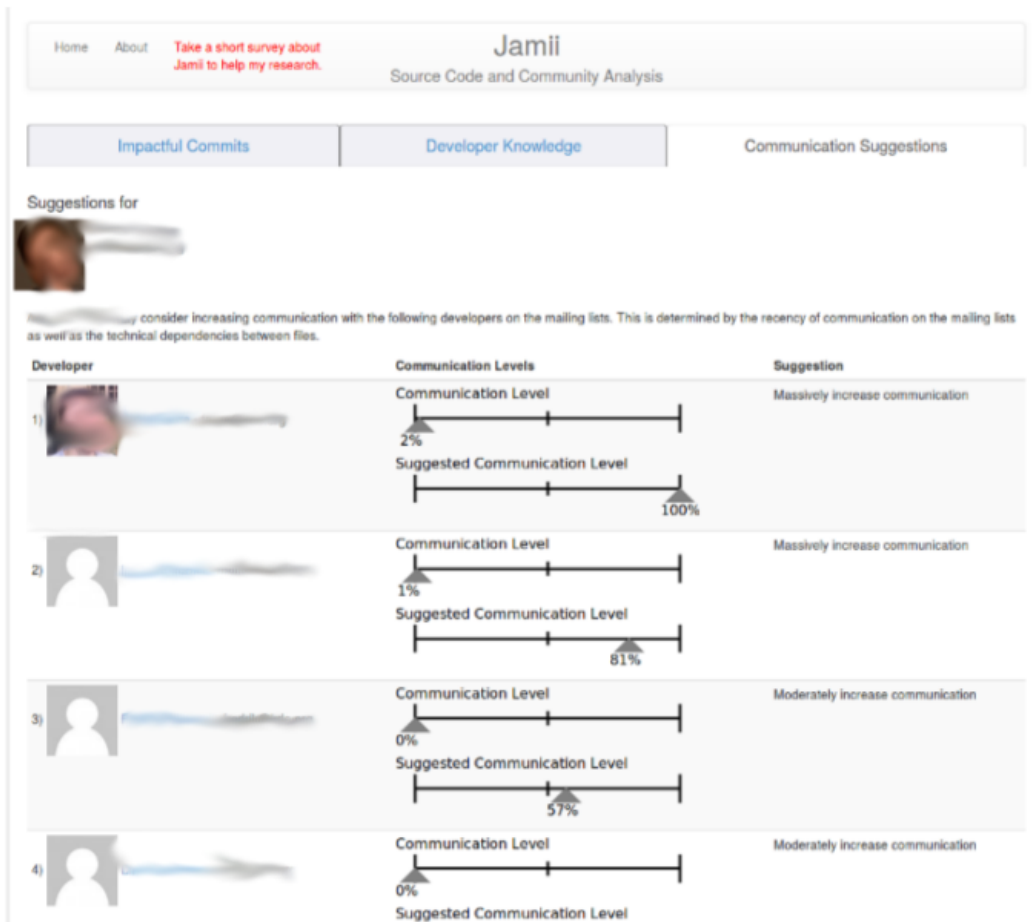
Strongly Disagree | | Strongly Agree

The "developer knowledge" section on the Jamii website would change the process I use to seek help and ask questions in the developer community.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "developer knowledge" questions below. e.g. What led you to answer as you did?

Use this reminder image of the Jamii website to answer the questions below.



The "communication suggestions" section on the Jamii website would increase my confidence in communicating with the appropriate people on the mailing list.

Strongly Disagree | | Strongly Agree

The "communication suggestions" section on the Jamii website would change the process I use to communicate on the mailing list.

Strongly Disagree | | Strongly Agree

The "communication suggestions" section on the Jamii website would change the process I use to communicate using other methods such as IRC.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above "communication suggestions" questions below. e.g. What led you to answer as you did?

The Jamii website in general would increase my integration and involvement in the community.

Strongly Disagree | | Strongly Agree

The Jamii website in general would increase my understanding of the overall codebase.

Strongly Disagree | | Strongly Agree

The Jamii website in general would result in an increase in the number of new developers in the community.

Strongly Disagree | | Strongly Agree

The Jamii website in general would increase the retention of existing developers in the community.

Strongly Disagree | | Strongly Agree

Please type comments or clarifications regarding the above questions below. e.g. What led you to answer as you did?

What is the email address that you use for KDE? The email address(es) you provide (optional) will be used to link this survey data to publicly available data of KDE such as mailing list posts, bug reports, and source code commits. If you use multiple email addresses, please list all of them. We will NOT distribute this email address to any other group.

I'm conducting one-on-one interviews via instant chat with people who have used Jamii. This is completely optional. Would you be willing to have me email you to setup a time for an interview?

- Yes
 No

If you have any questions, comments, or concerns please list them below. Thank you again and feel free to contact me via email for any questions or details: carlsonp@iastate.edu

BIBLIOGRAPHY

- Ada Initiative. (2015). *Ada initiative*. Internet: <http://adainitiative.org>. Retrieved from <http://adainitiative.org>
- ALERT. (2015). *Active support and real-time coordination based on event processing in floss development*. Internet: <http://www.alert-project.eu>. Retrieved from <http://www.alert-project.eu>
- Amrit, C., Hillegersberg, J., & Kumar, K. (2004). A Social Network perspective of Conway's Law. *Computer Supported Cooperative Work*, 1–10.
- Awstats. (2015). *Awstats*. Internet: <http://www.awstats.org>. Retrieved from <http://www.awstats.org>
- Awstats Documentation. (2015). *Awstats documentation*. Internet: http://www.awstats.org/docs/awstats_glossary.html. Retrieved from http://www.awstats.org/docs/awstats_glossary.html
- Beede, D., Julian, T., Langdon, D., McKittrick, G., Khan, B., & Doms, M. (2011). *Women in STEM: A Gender Gap to Innovation* (Tech. Rep.). U.S. Department of Commerce. Retrieved from <http://www.ssrn.com/abstract=1964782> doi: 10.2139/ssrn.1964782
- Begel, A., Phang, K. Y., & Zimmermann, T. (2010). Codebook: Discovering and Exploiting Relationships in Software Repositories. In *32nd international conference on software engineering* (pp. 125–134).
- Berger, C. R., & Calabrese, R. J. (1975). Some explorations in initial interaction and beyond: Toward a developmental theory of interpersonal communication. *Human Communication Research*, 99–112.
- Biehl, J. T., Czerwinski, M., Smith, G., & Robertson, G. G. (2007). FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. In *Proceedings of the sigchi*

- conference on human factors in computing systems* (pp. 1313–1322). Retrieved from <https://dl.acm.org/citation.cfm?id=1240823>
- Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2011). Don't Touch My Code!: Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th acm sigsoft symposium and the 13th european conference on foundations of software engineering* (pp. 4–14). Retrieved from <https://dl.acm.org/citation.cfm?id=2025119>
- Bitergia. (2015). *Bitergia*. Internet: <http://bitergia.com>. Retrieved from <http://bitergia.com>
- Bootstrap. (2015). *Bootstrap*. Internet: <http://getbootstrap.com>. Retrieved from <http://getbootstrap.com>
- Borici, A., Blincoe, K., Schröter, A., Valetto, G., & Damian, D. (2012). ProxiScientia: Toward Real-Time Visualization of Task and Developer Dependencies in Collaborating Software Development Teams. In *International conference on software engineering* (pp. 5–11).
- Burke, M., & Kraut, R. E. (2008). Mind Your Ps and Qs: The Impact of Politeness and Rudeness in Online Communities. In *Acm cscw 2008: Conference on computer supported cooperative work* (pp. 281–284).
- Burke, M., Kraut, R. E., & Joyce, E. (2010, December). Membership Claims and Requests: Conversation-Level Newcomer Socialization Strategies in Online Groups. *Small Group Research*, 41(1), 4–40. Retrieved from <http://sgr.sagepub.com/cgi/doi/10.1177/1046496409351936> doi: 10.1177/1046496409351936
- Burke, M., Marlow, C., & Lento, T. (2009). Feed Me: Motivating Newcomer Contribution in Social Network Sites. In *Acm chi 2009: Conference on computer-supported cooperative work* (pp. 945–954).
- Burke, M., Marlow, C., & Lento, T. (2010). Social network activity and social well-being. In *Acm chi 2010: Conference on human factors in computing systems* (pp. 1909–1912). Retrieved from <http://portal.acm.org/citation.cfm?doid=1753326.1753613> doi: 10.1145/1753326.1753613

- Cacioppo, J. T., Petty, R. E., Feinstein, J. A., & Jarvis, W. B. G. (1996). Dispositional Differences in Cognitive Motivation: The Life and Times of Individuals Varying in Need for Cognition. *Psychological Bulletin*, *119*(2), 197–253. Retrieved from <http://psycnet.apa.org/journals/bul/119/2/197/> doi: 10.1037/0033-2909.119.2.197
- Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008). Socio-technical congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development. In *Proceedings of the second acm-ieee international symposium on empirical software engineering and measurement - esem '08* (pp. 2–11). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=1414004.1414008> doi: 10.1145/1414004.1414008
- Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., & Carley, K. M. (2006). Identification of coordination requirements. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*, 353. Retrieved from <http://portal.acm.org/citation.cfm?doid=1180875.1180929> doi: 10.1145/1180875.1180929
- Ceruzzi, P. E. (2003). *A History of Modern Computing* (2nd ed.). The MIT Press.
- Chen, G., & Klimoski, R. (2003). The Impact of Expectations on Newcomer Performance in Teams as Mediated by Work Characteristics, Social Exchanges, and Empowerment. *Academy of Management*, *46*(5), 591–607.
- Clang. (2015). *Clang*. Internet: <http://clang.llvm.org>. Retrieved from <http://clang.llvm.org>
- CLOC. (2015). *Cloc*. Internet: <http://cloc.sourceforge.net>. Retrieved from <http://cloc.sourceforge.net>
- Collier, B., Burke, M., Kittur, N., & Kraut, R. E. (2010). Promoting Good Management: Governance, Promotion, and Leadership in Open Collaboration Communities. In *International conference on information systems (icis)* (pp. 1–10).
- Collins-Sussman, B., & Fitzpatrick, B. (2007). *How to Protect Your Open Source Project From Poisonous People*.
- Conway, M. E. (1968). How Do Committees Invent? *Datamation*, 1–12.

- Crowston, K., & Howison, J. (2005). The social structure of Free and Open Source software development. *First Monday*, 10(October), 1–21. Retrieved from <http://crowston.syr.edu/content/social-structure-free-and-open-source-software-development>
- Cubranic, D., & Murphy, G. C. (2003). Hipikat: recommending pertinent software development artifacts. *25th International Conference on Software Engineering, 2003. Proceedings.*, 6, 408–418. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1201219> doi: 10.1109/ICSE.2003.1201219
- Cubranic, D., Murphy, G. C., Singer, J., & Booth, K. (2005, June). Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering*, 31(6), 446–465. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463229> doi: 10.1109/TSE.2005.71
- D3.js. (2015). *D3.js*. Internet: <http://d3js.org>. Retrieved from <http://d3js.org>
- Dagenais, B., Ossher, H., Bellamy, R. K. E., Robillard, M. P., & de Vries, J. P. (2010). Moving into a new software project landscape. In *Proceedings of the 32nd acm/ieee international conference on software engineering - icse '10* (Vol. 1, p. 275). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doi=1806799.1806842> doi: 10.1145/1806799.1806842
- Daniel, S., Agarwal, R., & Stewart, K. J. (2012, August). The Effects of Diversity in Global, Distributed Collectives: A Study of Open Source Project Success. *Information Systems Research*, 1–22. Retrieved from <http://isr.journal.informs.org/cgi/doi/10.1287/isre.1120.0435> doi: 10.1287/isre.1120.0435
- da Silva, I. A., Chen, P. H., der Westhuizen, C. V., Ripley, R. M., & van der Hoek, A. (2006). Lighthouse: Coordination through Emerging Design. In *Proceedings of the 2006 oopsla workshop on eclipse technology exchange* (pp. 11–15).
- Datatables. (2015). *Datatables*. Internet: <https://www.datatables.net>. Retrieved from <https://www.datatables.net>
- David, P. A., Waterman, A., & Arora, S. (2003). *The Free/Libre/Open Source Software Survey for 2003* (Tech. Rep.). Stanford Institute for Economic Policy Research. Retrieved from <http://stanford.edu/group/floss-us/>

- Debian. (2015). *Debian social contract*. Internet: http://debian.org/social_contract. Retrieved from http://debian.org/social_contract
- Debian Women. (2015). *Debian women*. Internet: <http://women.debian.org>. Retrieved from <http://women.debian.org>
- Deshpande, A., & Riehle, D. (2008). The Total Growth of Open Source. In *Proceedings of the fourth conference on open source systems* (pp. 197–209).
- de Souza, C. R., Froehlich, J., & Dourish, P. (2005). Seeking the Source: Software Source Code as a Social and Technical Artifact. *Conference on Supporting Group Work*, 197–206.
- de Souza, C. R., Quirk, S., Trainer, E., & Redmiles, D. F. (2007). Supporting collaborative software development through the visualization of socio-technical dependencies. *Proceedings of the 2007 international ACM conference on Conference on supporting group work - GROUP '07*, 147. Retrieved from <http://portal.acm.org/citation.cfm?doid=1316624.1316646> doi: 10.1145/1316624.1316646
- Dev Chix. (2015). *Dev chix*. Internet: <http://www.devchix.com>. Retrieved from <http://www.devchix.com>
- Dewan, P., & Hegde, R. (2007). Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. In *Proceedings of the tenth european conference on computer supported cooperative work* (pp. 24–28). Retrieved from <http://www.springerlink.com/content/x58r16x73p670554/>
- Django. (2015). *Django*. Internet: <https://www.djangoproject.com>. Retrieved from <https://www.djangoproject.com>
- Dubinsky, Y., & Hazzan, O. (2008). Using Leadership to Analyze Socio-Technical Congruence. In *Workshop on socio-technical congruence (stc)*.
- Dubrovsky, V., Kiesler, S., & Sethna, B. (1991). The Equalization Phenomenon: Status Effects in Computer-Mediated and Face-to-Face Decision-Making Groups. *Human Computer Interaction*, 6, 119–146.
- Ducheneaut, N. (2003). *The reproduction of Open Source Software programming communities* (PhD thesis). University of California, Berkeley.

- Ducheneaut, N. (2005, July). Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4), 323–368. Retrieved from <http://www.springerlink.com/index/10.1007/s10606-005-9000-1> doi: 10.1007/s10606-005-9000-1
- Dyer, R., Nguyen, H., Rajan, H., & Nguyen, T. N. (2012). Analyzing ultra-large-scale code corpus with boa. In *Proceedings of the 3rd annual conference on systems, programming, and applications: software for humanity - splash '12* (pp. 25–26). ACM Press. Retrieved from <http://dl.acm.org/citation.cfm?doid=2384716.2384729> doi: 10.1145/2384716.2384729
- Ehrlich, K., Helander, M. E., Valetto, G., Williams, C., & Davies, S. (2008). An Analysis of Congruence Gaps and Their Effect on Distributed Software Development. *international workshop on socio-technical congruence (STC 2008)*, 1–10.
- Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., & Mockus, A. (2001). Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Transactions on Software Engineering*, 27(1), 1–12. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=895984&tag=1
- Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007a, July). Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development. *Organization Science*, 18(4), 613–630. Retrieved from <http://orgsci.journal.informs.org/cgi/doi/10.1287/orsc.1070.0297> doi: 10.1287/orsc.1070.0297
- Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007b, July). Team Knowledge and Coordination in Geographically Distributed Software Development. *Journal of Management Information Systems*, 24(1), 135–169. Retrieved from <http://mesharpe.metapress.com/openurl.asp?genre=article&id=doi:10.2753/MIS0742-1222240104> doi: 10.2753/MIS0742-1222240104
- Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2), 115–150. doi: 10.1109/ICSE.2000.870431

- Figueiredo, M. C., & de Souza, C. R. (2012). Wolf: Supporting Impact Analysis Activities in Distributed Software Development. In *International conference on software engineering* (pp. 40–46).
- Finkelstein, S. R., & Fishbach, A. (2012, June). Tell Me What I Did Wrong: Experts Seek and Respond to Negative Feedback. *Journal of Consumer Research*, *39*(1), 22–38. Retrieved from <http://www.jstor.org/stable/info/10.1086/661934> doi: 10.1086/661934
- Fischer, G. (2011, June). Understanding, Fostering, and Supporting Cultures of Participation. *Interactions*, *80*, 42–53.
- Fogel, K. (2005). *Producing Open Source Software How to Run a Successful Free Software Project*. O'Reilly Media.
- Free Software Foundation. (2015). *Free software definition*. Internet: <https://www.gnu.org/philosophy/free-sw.html>. Retrieved from <https://www.gnu.org/philosophy/free-sw.html>
- Fritz, T., Murphy, G. C., & Hill, E. (2007). Does a Programmer's Activity Indicate Knowledge of Code? In *Proceedings of the 6th join meeting of the european software engineering conference and the acm sigsoft symposium on the foundations of software engineering* (pp. 341–350). Retrieved from <https://dl.acm.org/citation.cfm?id=1287673>
- Fritz, T., Ou, J., Murphy, G. C., & Murphy-Hill, E. (2010). A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd acm/ieee international conference on software engineering - icse '10* (Vol. 1, p. 385). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=1806799.1806856> doi: 10.1145/1806799.1806856
- Frost, R. (2007). Jazz and the Eclipse Way of Collaboration. *IEEE Software*, *24*(6), 114–117. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4375254&tag=1
- Ghosh, R. A., Glott, R., Krieger, B., & Robles, G. (2002). *Free/Libre and Open Source Software: Survey and Study* (Tech. Rep. No. June). Maastricht, Netherlands: University of Maastricht.

- Gilbert, E., & Karahalios, K. (2007). CodeSaw: A Social Visualization of Distributed Software Development. *INTERACT'07: Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction*, 303–316.
- Girba, T., Kuhn, A., Seeberger, M., & Stéphane, D. (2005). How Developers Drive Software Evolution. In *Eighth international workshop on principles of software evolution* (pp. 113–122).
- Girba, T., Stéphane, D., & Lanza, M. (2004). Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes. In *20th ieee international conference on software maintenance* (pp. 40–49). Retrieved from https://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1357788&tag=1
- Girl Develop It. (2015). *Girl develop it*. Internet: <https://www.girldevelopit.com>. Retrieved from <https://www.girldevelopit.com>
- Git. (2015). *Git*. Internet: <http://git-scm.com>. Retrieved from <http://git-scm.com>
- Github. (2015). *Github*. Internet: <http://www.github.com>. Retrieved from <http://www.github.com>
- Gmane. (2015). *Gmane*. Internet: <http://www.gmane.org>. Retrieved from <http://www.gmane.org>
- Gonzalez-Barahona, J. M., López, L., Robles, G., Sistemas, G., & Juan Carlos, R. (2004). Community structure of modules in the Apache project. In *"collaboration, conflict and control: The 4th workshop on open source software engineering" w8s workshop - 26th international conference on software engineering* (pp. 43–47). Iee. Retrieved from <http://link.aip.org/link/IEESEM/v2004/i908/p43/s1&Agg=doi> doi: 10.1049/ic:20040263
- Gould, P. R. (1967). On the Geographical Interpretation of Eigenvalues. *Transactions of the Institute of British Geographers*, Dec. 1967(42), 53–86. Retrieved from <http://www.jstor.org/stable/621372>
- Gravatar. (2015). *Gravatar*. Internet: <https://en.gravatar.com>. Retrieved from <https://en.gravatar.com>
- Hattori, L., & Lanza, M. (2009, May). Mining the history of synchronous changes to refine code ownership. In *2009 6th ieee international working conference on mining software*

- repositories* (pp. 141–150). Ieee. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5069492> doi: 10.1109/MSR.2009.5069492
- Hattori, L., & Lanza, M. (2010). Syde: A Tool for Collaborative Software Development. In *32nd international conference on software engineering* (pp. 235–238). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6062168
- Hendin, H. M., & Cheek, J. M. (1997). Assessing Hypersensitive Narcissism: A Reexamination of Murray's Narcism Scale. *Journal of Research in Personality*, *31*, 588–599.
- Herbsleb, J. D., & Grinter, R. E. (1999). Splitting the organization and integrating the code. *Proceedings of the 21st international conference on Software engineering - ICSE '99*, 85–95. Retrieved from <http://portal.acm.org/citation.cfm?doid=302405.302455> doi: 10.1145/302405.302455
- Herbsleb, J. D., Roberts, J. A., & Mockus, A. (2006). Collaboration in Software Engineering Projects: A Theory of Coordination. *ICIS 2006 General Topics*, 1–16.
- Herbsleb, J. D., Sarma, A., Mockus, A., & Cataldo, M. (2008, April). Using Distributed Constraint Satisfaction to Build a Theory of Congruence. In *Stc workshop 2008* (p. 4).
- Herraiz, I., Robles, G., Amor, J. J., Romera, T., & Gonzalez-barahona, J. M. (2006). The processes of joining in global distributed software projects. *Proceedings of the 2006 international workshop on Global software development for the practitioner - GSD '06*, 27. Retrieved from <http://portal.acm.org/citation.cfm?doid=1138506.1138513> doi: 10.1145/1138506.1138513
- Hertel, G., Niedner, S., & Herrmann, S. (2003, July). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, *32*(7), 1159–1177. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0048733303000477> doi: 10.1016/S0048-7333(03)00047-7
- Hinds, P. J., & Mcgrath, C. (2006). Structures that Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams. In *Proceedings of the 2006 20th anniversary conference on computer supported cooperative work* (pp. 343–352).

- Hollander, E. P. (1958, March). Conformity, status, and idiosyncrasy credit. *Psychological review*, *65*(2), 117–27. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/13542706>
- Holliger, A. (2007). *The Culture of Open Source Computing*.
- Holtgraves, T. (2010). Social psychology and language: Words, utterances, and conversations. In D. Gilbert, S. Fiske, & G. Lindzey (Eds.), *The handbook of social psychology* (5th editio ed., chap. 36). New York: McGraw-Hill.
- Horton, W., & Gerrig, R. J. (2005, July). Conversational Common Ground and Memory Processes in Language Production. *Discourse Processes*, *40*(1), 1–35. Retrieved from <http://www.informaworld.com/openurl?genre=article&doi=10.1207/s15326950dp4001.1&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3> doi: 10.1207/s15326950dp4001_1
- Ilgén, D. R., Hollenbeck, J. R., Johnson, M., & Jundt, D. (2005, January). Teams in organizations: from input-process-output models to IMO models. *Annual review of psychology*, *56*, 517–43. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/15709945> doi: 10.1146/annurev.psych.56.091103.070250
- Jamii Github. (2015). *Jamii github*. Internet: <http://github.com/carlsonp/jamii>. Retrieved from <http://github.com/carlsonp/jamii>
- John, O. P., & Srivastava, S. (1999). The Big Five Trait Taxonomy: History, Measurement, and Theoretical Perspectives. In *Handbook of personality: Theory and research* (2nd editio ed., pp. 102–138). New York: Guilford Press.
- jqPlot. (2015). *jqplot*. Internet: <http://www.jqplot.com>. Retrieved from <http://www.jqplot.com>
- jQuery. (2015). *jquery*. Internet: <https://jquery.com>. Retrieved from <https://jquery.com>
- Kagdi, H., Hammad, M., & Maletic, J. I. (2008). Who Can Help Me with this Source Code Change? In *Ieee international conference on software maintenance (icsm)* (pp. 157–166). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4658064&tag=1
- KDE e.V. (2015). *Kde e.v*. Internet: <http://ev.kde.org>. Retrieved from <http://ev.kde.org>

- KDE Getting Started. (2015). *Kde getting started*. Internet: https://techbase.kde.org/Getting_Started/Sources. Retrieved from https://techbase.kde.org/Getting_Started/Sources
- Krackhardt, D., & Carley, K. M. (1998). A PCANS Model of Structure in Organizations. In *Proceedings of the 1998 international symposium on command and control research and technology* (pp. 113–119).
- Kraut, R. E., Burke, M., & Riedl, J. (2011). Dealing with Newcomers. In *Evidence-based social design: Mining the social sciences to build online communities* (pp. 1–42). MIT Press.
- Kwan, I. (2011). *The Study of Socio-technical Coordination Using a Socio-technical Congruence Model* (PhD thesis). University of Victoria.
- Kwan, I., Schr, A., & Damian, D. (2009). A Weighted Congruence Measure. *international workshop on socio-technical congruence (STC 2009)*, 4.
- Kwan, I., Schröter, A., & Damian, D. (2011). Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *IEEE Transactions on Software Engineering*, 37(3), 307–324.
- Lakhani, K. R., & Wolf, R. G. (2003). *Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects*. Retrieved from <http://dx.doi.org/10.2139/ssrn.443040>
- Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.
- Leary, M. R., Kelly, K. M., Cottrell, C. A., & Schreindorfer, L. S. (2007). *Individual differences in the need to belong: Mapping the nomological network* (No. 2007). Duke University.
- Leathers, D. G. (1972). Quality of group communication as a determinant of group product. *Speech Monographs*, 39(3), 166–173. Retrieved from <http://www.tandfonline.com/doi/abs/10.1080/03637757209375754> doi: 10.1080/03637757209375754
- Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., . . . Wolff, S. (2009). A Brief History of the Internet. *ACM SIGCOMM Computer Communication Review*, 39(5), 22–31.

- Lewis, K. (2003). Measuring transactive memory systems in the field: Scale development and validation. *Journal of Applied Psychology, 88*(4), 587–604. Retrieved from <http://doi.apa.org/getdoi.cfm?doi=10.1037/0021-9010.88.4.587> doi: 10.1037/0021-9010.88.4.587
- Linux Chix. (2015). *Linux chix*. Internet: <http://www.linuxchix.org>. Retrieved from <http://www.linuxchix.org>
- LLVM. (2015). *Llvm*. Internet: <http://llvm.org>. Retrieved from <http://llvm.org>
- Madey, G., Freech, V., & Tynan, R. (2004). *Modeling the Free/Open Source Software Community: A Quantitative Investigation*.
- McCroskey, J. C. (1992). Reliability and Validity of the Willingness to Communicate Scale. *Communication, 40*(1), 16–25.
- McDonald, D. W., & Ackerman, M. S. (2000). Expertise Recommender: A Flexible Recommendation System and Architecture. In *Proceedings of the 2000 acm conference on computer supported cooperative work* (pp. 231–240).
- McGrath, J. E. (1997). Small group research, that once and future field: An interpretation of the past with an eye to the future. *Group Dynamics: Theory, Research, and Practice, 1*(1), 7–27. doi: 10.1037/1089-2699.1.1.7
- McGrew, J. F., Bilotta, J. G., & Deeney, J. M. (1999, April). Software Team Formation and Decay: Extending the Standard Model for Small Groups. *Small Group Research, 30*(2), 209–234. Retrieved from <http://sgr.sagepub.com/cgi/doi/10.1177/104649649903000204> doi:10.1177/104649649903000204
- McGuire, T. W., Kiesler, S., & Siegel, J. (1987). Group and computer-mediated discussion effects in risk decision making. *Journal of Personality and Social Psychology, 52*(5), 917–930. Retrieved from <http://doi.apa.org/getdoi.cfm?doi=10.1037/0022-3514.52.5.917> doi: 10.1037/0022-3514.52.5.917
- McLuhan, M. (1964). *Understanding Media: The Extensions of Man*. New York: McGraw Hill.
- Milgram, S. (1967). The Small World Problem. *Psychology Today, 1*(1), 61–67.

- Miller, K. (1998). *Organizational communication approaches and processes* (Second Edi ed.). Wadsworth Publishing Company.
- Minto, S., & Murphy, G. C. (2007, May). Recommending Emergent Teams. *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4228642> doi: 10.1109/MSR.2007.27
- Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002, July). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3). Retrieved from <http://portal.acm.org/citation.cfm?doid=567793.567795> doi: 10.1145/567793.567795
- Mockus, A., & Herbsleb, J. D. (2002). Expertise Browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th international conference on software engineering. icse 2002* (pp. 503–512). ACM. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1007994> doi: 10.1109/ICSE.2002.1007994
- MongoDB. (2015). *Mongodb*. Internet: <https://www.mongodb.org>. Retrieved from <https://www.mongodb.org>
- Moon, J. Y., & Sproull, L. (2008). Essence of Distributed Work: The Case of the Linux Kernel. *First Monday*, 5(11), 1–21. Retrieved from http://www.firstmonday.org/issues/issue5_11/moon/index.html
- Mozilla Foundation. (2015). *Mozilla foundation*. Internet: <https://www.mozilla.org/en-US/foundation/>. Retrieved from <https://www.mozilla.org/en-US/foundation/>
- Nafus, D., Leach, J., & Krieger, B. (2006). *Free/Libre and Open Source Software: Policy Support* (Vol. 16) (No. March).
- National Center for Education Statistics. (2015). *National center for education statistics*. Internet: http://nces.ed.gov/programs/digest/2014menu_tables.asp. Retrieved from http://nces.ed.gov/programs/digest/2014menu_tables.asp
- Neo4j. (2015). *Neo4j graph database*. Internet: <http://www.neo4j.org>. Retrieved from <http://www.neo4j.org>

- Neo4j Cypher Refcard. (2015). *Neo4j cypher refcard*. Internet: <http://neo4j.com/docs/2.0/cypher-refcard/>. Retrieved from <http://neo4j.com/docs/2.0/cypher-refcard/>
- Nguyen, T. T., Nguyen, T. N., Duesterwald, E., Klinger, T., & Santhanam, P. (2012). Inferring Developer Expertise through Defect Analysis. In *International conference on software engineering* (pp. 1297–1300).
- Nonnecke, B., & Preece, J. (2000). Lurker demographics: Counting the silent. In *Proceedings of sigchi conference on human factors in computing systems* (pp. 73–80).
- Ohlsson, M. C., Mayrhauser, A. V., Mcguire, B., & Wohlin, C. (1999). Code Decay Analysis of Legacy Software through Successive Releases. In *Ieee aerospace conference proceedings* (pp. 69–81). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=790190
- Open Source Initiative. (2015). *Open source initiative*. Internet: <http://opensource.org/osd>. Retrieved from <http://opensource.org/osd>
- Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). *The PageRank Citation Ranking: Bringing Order to the Web* (Tech. Rep.). Retrieved from <http://ilpubs.stanford.edu:8090/422>
- Palazzolo, E. T., Serb, D. A., She, Y., Su, C., & Contractor, N. S. (2006, May). Coevolution of Communication and Knowledge Networks in Transactive Memory Systems: Using Computational Models for Theoretical Development. *Communication Theory*, *16*(2), 223–250. Retrieved from <http://doi.wiley.com/10.1111/j.1468-2885.2006.00269.x>
doi: 10.1111/j.1468-2885.2006.00269.x
- Parnas, D. L. (1972, December). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058. Retrieved from <http://portal.acm.org/citation.cfm?doid=361598.361623> doi: 10.1145/361598.361623
- Pennebaker, J. W., Chung, C. K., Ireland, M., Gonzales, A., & Booth, R. J. (2007). *The Development and Psychometric Properties of LIWC2007*.
- Pfeffer, J. (1983). Organisational Demography. In L. L. Cummings & B. M. Staw (Eds.), *Research in organisational behavior* (pp. 299–357). Greenwich, Conn: JAI Press.

- Powell, W. E., Hunsinger, D. S., & Medlin, B. D. (2010). Gender Differences Within the Open Source Community: An Exploratory Study. *Journal of Information Technology Management, XXI*(4), 29–37.
- Pygments. (2015). *Pygments*. Internet: <http://pygments.org>. Retrieved from <http://pygments.org>
- Python. (2015). *Python*. Internet: <https://www.python.org>. Retrieved from <https://www.python.org>
- R Project. (2015). *R project*. Internet: <http://www.r-project.org>. Retrieved from <http://www.r-project.org>
- Raymond, E. S. (1999, September). The cathedral and the bazaar. *Knowledge, Technology & Policy, 12*(3), 23–49. Retrieved from <http://www.springerlink.com/index/10.1007/s12130-999-1026-0> doi: 10.1007/s12130-999-1026-0
- Reddit. (2015). *Reddit*. Internet: <http://www.reddit.com>. Retrieved from <http://www.reddit.com>
- Richard Stallman. (2015a). *Free software foundation*. Internet: <http://www.fsf.org>. Retrieved from <http://www.fsf.org>
- Richard Stallman. (2015b). *Gnu operating system*. Internet: <http://www.gnu.org>. Retrieved from <http://www.gnu.org>
- Ridings, C., Gefen, D., & Arinze, B. (2006). Psychological Barriers: Lurker and Poster Motivation and Behavior in Online Communities. *Communications of the Association for Information Systems, 18*(1), 329–354.
- Rigby, P. C., & Hassan, A. E. (2007, May). What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, 23–23. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4228660> doi: 10.1109/MSR.2007.35
- Roberts, J. A., Hamm, I.-H., & Slaughter, S. A. (2006, July). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science, 52*(7), 984–999. Retrieved

- from <http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.1060.0554> doi: 10.1287/mnsc.1060.0554
- Robles, G., & Gonzalez-Barahona, J. M. (2006). Contributor Turnover in Libre Software Projects. In E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, & G. Succi (Eds.), *Open source systems* (Vol. 203, pp. 273–286). Springer Boston. Retrieved from http://dx.doi.org/10.1007/0-387-34226-5_28 doi: http://dx.doi.org/10.1007/0-387-34226-5_28
- Robles, G., Koch, S., & González-Barahona, J. M. (2004). Remote analysis and measurement of libre software systems by means of the CVSanaly tool. In *Proceedings of the second international workshop on remote analysis and measurement of software systems (ramss 04) - w15s workshop - 26th international conference on software engineering* (pp. 51–55). Retrieved from [http://digital-library.theiet.org/content/conferences/10.1049/ic_20040351\\$delimiter"026E30F\\$nhhttp://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.6959](http://digital-library.theiet.org/content/conferences/10.1049/ic_20040351$delimiter) doi: 10.1049/ic:20040351
- Rodríguez, J. P., Ebert, C., & Vizcaino, A. (2010). Technologies and Tools for Distributed Teams. *IEEE Software*, 27(5), 10–14. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5551012
- Rusbult, C. E., Martz, J. M., & Agnew, C. R. (1998, December). The Investment Model Scale: Measuring commitment level, satisfaction level, quality of alternatives, and investment size. *Personal Relationships*, 5(4), 357–387. Retrieved from <http://doi.wiley.com/10.1111/j.1475-6811.1998.tb00177.x> doi: 10.1111/j.1475-6811.1998.tb00177.x
- Sack, W., Détienne, F., Ducheneaut, N., Burkhardt, J.-M., Mahendran, D., & Barcellini, F. (2006, June). A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software. *Computer Supported Cooperative Work (CSCW)*, 15(2-3), 229–250. Retrieved from <http://www.springerlink.com/index/10.1007/s10606-006-9020-5http://www2.parc.com/csl/members/nicolas/browser.html> doi: 10.1007/s10606-006-9020-5
- Sarma, A., & Herbsleb, J. (2008b). Exploring Interrelationships among Project Entities to Support Coordination in Distributed Teams. In *Workshop on supporting distributed team work at cscw 2008*.

- Sarma, A., & Herbsleb, J. D. (2008a). Challenges in Measuring, Understanding, and Achieving Social-Technical Congruence. *CMU Technical Report*.
- Sarma, A., & Herbsleb, J. D. (2008c). Using Development Experience to Calculate Congruence. *international workshop on socio-technical congruence (STC 2008)*.
- Sarma, A., Maccherone, L., Wagstrom, P. A., & Herbsleb, J. D. (2009, May). Tesseract: Interactive visual exploration of socio-technical relationships in software development. *2009 IEEE 31st International Conference on Software Engineering*, 23–33. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5070505> doi: 10.1109/ICSE.2009.5070505
- Sarma, A., Noroozi, Z., & van der Hoek, A. (2003). Palantir: raising awareness among configuration management workspaces. *25th International Conference on Software Engineering, 2003. Proceedings.*, 444–454. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1201222> doi: 10.1109/ICSE.2003.1201222
- Sarma, A., Redmiles, D., & van der Hoek, A. (2008). Empirical evidence of the benefits of workspace awareness in software configuration management. In *Proceedings of the 16th acm sigsoft international symposium on foundations of software engineering - sigsoft '08/fse-16* (pp. 113–123). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=1453101.1453118> doi: 10.1145/1453101.1453118
- Sarma, A., Redmiles, D. F., & van der Hoek, A. (2012). Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering*, 38(4), 889–908.
- Scacchi, W. (2004). Free and Open Source Development Practices in the Game Community. *Ieee Software*, 59–66.
- Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Methods. *Advances in Computers*, 69, 243–295.
- Schneider, K. A., Gutwin, C., Penner, R., & Paquette, D. (2004). Mining a Software Developer's Local Interaction History. In *Proceedings of msr 2004 (1st international workshop on mining software repositories)* (pp. 5–9).

- Schümmer, T., & Haake, J. M. (2001). Supporting Distributed Software Development by Modes of Collaboration. In *Proceedings of the seventh european conference on computer supported cooperative work* (pp. 79–98). Retrieved from <https://dl.acm.org/citation.cfm?id=1241872>
- Shah, S. K. (2006, July). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, *52*(7), 1000–1014. Retrieved from <http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.1060.0553> doi: 10.1287/mnsc.1060.0553
- Sharma, V. S., & Kaulgud, V. (2012). PIVoT: Project Insights and Visualization Toolkit. In *International conference on software engineering* (pp. 63–69).
- Shestowsky, D., Wegener, D. T., & Fabrigar, L. R. (1998). Need for cognition and interpersonal influence: Individual differences in impact on dyadic decisions. *Journal of Personality and Social Psychology*, *74*(5), 1317–1328. Retrieved from <http://doi.apa.org/getdoi.cfm?doi=10.1037/0022-3514.74.5.1317> doi: 10.1037/0022-3514.74.5.1317
- Sllinen-Kuparinen, A., McCroskey, J. C., & Richmond, V. (1991). Willingness to communicate, communication apprehension, introversion, and self-reported communication competence: Finnish and American comparisons. In *World communication association* (p. 22).
- Spinellis, D., & Giannikas, V. (2012, March). Organizational adoption of open source software. *Journal of Systems and Software*, *85*(3), 666–682. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0164121211002512> doi: 10.1016/j.jss.2011.09.037
- Sproull, L., & Kiesler, S. (1986, November). Reducing Social Context Cues: Electronic Mail in Organizational Communication. *Management Science*, *32*(11), 1492–1512. Retrieved from <http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.32.11.1492> doi: 10.1287/mnsc.32.11.1492
- Stallman, R. (1985). *The GNU Manifesto*. Retrieved 2012-11-04, from <https://www.gnu.org/gnu/manifesto.html>
- Stephany, F., Mens, T., & Gırba, T. (2009). Maispion: A Tool for Analysing and Visualising Open Source Software Developer Communities. In *Proceedings of international workshop*

- on *smalltalk technologies (iwst 2009)* (pp. 50–57). ACM. Retrieved from <https://dl.acm.org/citation.cfm?doid=1735935.1735944> doi: 1735935.1735944
- Straffin, P. D. (1980, November). Linear Algebra in Geography: Eigenvectors of Networks. *Mathematics Magazine*, 53(5), 269. Retrieved from <http://www.jstor.org/stable/10.2307/2689388?origin=crossref> doi: 10.2307/2689388
- Strohmaier, M., & Wermelinger, M. (2009). Using Network Properties to Study Congruence of Software Dependencies and Maintenance Activities in Eclipse. In *2nd international workshop on socio-technical congruence* (p. 4).
- Tango Desktop Project. (2015). *Tango desktop project*. Internet: <http://tango.freedesktop.org>. Retrieved from <http://tango.freedesktop.org>
- Titan. (2015). *Titan graph database*. Internet: <https://github.com/thinkaurelius/titan>. Retrieved from <https://github.com/thinkaurelius/titan>
- Torvalds, L., & Diamond, D. (2002). *Just for Fun: The Story of an Accidental Revolutionary*. Harper Paperbacks.
- Tuomi, I. (2000). Internet, Innovation, and Open Source: Actors in the Network. *The Association of Internet Researchers Conference*, 1–34.
- typeahead.js. (2015). *typeahead.js*. Internet: <https://twitter.github.io/typeahead.js>. Retrieved from <https://twitter.github.io/typeahead.js>
- Ubuntu Women. (2015). *Ubuntu women*. Internet: <http://wiki.ubuntu-women.org>. Retrieved from <http://wiki.ubuntu-women.org>
- Valetto, G., Helander, M. E., Ehrlich, K., Chulani, S., Wegman, M., & Williams, C. (2007, May). Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, 25. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4228662> doi: 10.1109/MSR.2007.33
- Valetto, G., Jose, S., & Williams, C. (2008). Balancing the Value and Risk of Socio-Technical Congruence. In *proceedings of the icse workshop on socio-technical congruence (stc 2008)* (p. 4).

- von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003, July). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7), 1217–1241. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0048733303000507> doi: 10.1016/S0048-7333(03)00050-7
- Wagstrom, P. A., & Herbsleb, J. D. (2008). Individualized Socio-Technical Congruence. In *Workshop on socio-technical congruence at icse 2008* (pp. 1–4).
- Wagstrom, P. A., Herbsleb, J. D., & Carley, K. M. (2009). Decaying Socio-Technical Congruence as a Method to Account for Architectural Changes. In *international workshop on socio-technical congruence (stc 2009)*.
- Walli, S., Gynn, D., & Rotz, B. V. (2005, April). *The Growth of Open Source Software in Organizations* (Vol. 141; Tech. Rep. No. 4). Boston, MA: Optaros Inc.
- Weber, S. (2004). *The Success of Open Source*. Harvard University Press.
- Wegner, D. M., Erber, R., & Raymond, P. (1991, December). Transactive memory in close relationships. *Journal of personality and social psychology*, 61(6), 923–929. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/1774630>
- Wenger, E., McDermott, R., & Snyder, W. M. (2002). *Cultivating Communities of Practice*. Harvard Business Press.
- Williams, S. (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly. Retrieved from <http://oreilly.com/openbook/freedom/>
- Women in Drupal. (2015). *Women in drupal (formerly drupalchix)*. Internet: <http://groups.drupal.org/drupalchix>. Retrieved from <http://groups.drupal.org/drupalchix>
- Xiang, P. F., Ying, A. T., Cheng, P., Dang, Y. B., Ehrlich, K., Helander, M. E., . . . Yang, S. X. (2008). Ensemble: a Recommendation Tool for Promoting Communication in Software Team. *Proceedings of the 2008 international workshop on Recommendation systems for software engineering - RSSE '08*, 2. Retrieved from <http://portal.acm.org/citation.cfm?doid=1454247.1454259> doi: 10.1145/1454247.1454259
- Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. (2000). Collaboration with Lean Media: How Open-Source Software Succeeds. In *Proceedings of the 2000 acm confer-*

ence on computer supported cooperative work - cscw '00 (pp. 329–338). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=358916.359004> doi: 10.1145/358916.359004

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. *25th International Conference on Software Engineering, 2003. Proceedings.*, 419–429. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1201220> doi: 10.1109/ICSE.2003.1201220