

**CONTEMPORARY PEER CODE REVIEW PRACTICES
AND ASSOCIATED BENEFITS**

by

AMMIANGSHU BOSU

DR. JEFFREY C. CARVER, COMMITTEE CHAIR

DR. JEFF GRAY

DR. MUNAWAR HAFIZ

DR. RANDY SMITH

DR. MARCUS BROWN

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2015

Copyright Amiangshu Bosu 2015
ALL RIGHTS RESERVED

ABSTRACT

Prior research indicates that peer code review is an effective method for reducing the number of defects and improving the quality of code. Besides maintaining the integrity of the code, code review spreads knowledge, expertise, and development techniques among the review participants. In the recent years, many Open Source Software (OSS) communities and commercial organizations have adopted 'contemporary' or 'modern' code review, an informal, regular, tool-based process. Because both OSS and commercial developers spend a significant amount of effort performing code reviews, the primary goal of this dissertation is to better understand contemporary code review practices, its benefits, and factors that influence its outcomes.

To address this goal, this dissertation describes empirical studies using surveys, software repository mining, and social network analysis. The first study is a survey of OSS developers to understand their collaboration and the process by which they form impressions of each other. The results suggest that coding-related factors influence impression formation the most among OSS developers. Therefore, the types of interactions where participants can judge a peers code or creativity (e.g., code review) should be crucial for peer impression formation. The results of this study motivated the selection of peer code review as the focus of this dissertation. The second study describes a survey of developers from 36 popular OSS projects and from Microsoft about: 1) the code review process in their projects, 2) their expectations from code review, and 3) how code review impacts impressions about their peers. The results suggest that the primary perceived benefit of code review is knowledge sharing, relationship building, better designs, and ensuring maintainable code, as opposed to the expected result of defect detection. Code reviews help build impressions between code review participants. Those impressions not only impact code review process but also future collaborations among developers. Due to the rarity of face-to-face interactions, OSS developers rely more on the reputation of and relationship with the author during code reviews. Conversely, Microsoft developers focus more on expertise and anticipated efforts.

Finally, the third study aims to find the impact of developers' reputation on the outcome of his/her code review requests. The results suggest that developers' reputations help them receive quicker feedback on their review requests, complete reviews in shorter time, and get their

code changes accepted. Newcomers to OSS projects suffer the most due to delayed feedback, which may discourage their future participation. A reviewer recommendation system to triage incoming code review requests can be useful to reduce delayed feedback for newcomers. Based on the results from these studies, this dissertation makes recommendations for practitioners to adopt and improve code review practices.

DEDICATION

To my parents

To my wife Mousumi

To rest of my family

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude and appreciation to my professor, mentor, and adviser, Dr. Jeffrey Carver. Without his guidance, support, and assistance it would not be possible for me to finish my dissertation. Dr. Carver always helped in identifying, formulating, and solving research problems, and patiently revised and re-revised the publications and presentations to improve the quality of my research results. I really appreciate his assistance during my job search, patiently pointing out my mistakes, and teaching me how to communicate better. His guidance helped me become a better researcher, a better professional, and overall a better person. Thank you, Dr. Carver, for motivating me and always being available to help me through this process.

To my other committee members, I am grateful for your willingness to take part in the evaluation of my dissertation research. Dr. Jeff Gray - I appreciate you for spending time to review my faculty application materials and sending recommendation letters. Dr. Munawar Hafiz - I appreciate you for providing important guidance in the security project, answering my phone calls / emails, and soliciting valuable career advice. Dr. Marcus Brown - I appreciate your suggestions and your efforts to point out the grammatical mistakes and typos in this dissertation. Dr. Randy Smith - I appreciate your suggestions in improving my dissertation. I would like to add a special thanks to Dr. Nicholas Kraft for advising me whenever I seek directions and providing me valuable support during my job search.

My sincerest gratitude and appreciation goes to my lovely wife Mousumi Das for constantly supporting me and going through hardships during my PhD years. I would also like to mention my mom, dad, and brother. They always provided me great support and motivation to pursue a PhD.

I am also privileged to be associated with the current software engineering group at UA and like to thank my fellow labmates: Debarshi Chatterji, Ferosh Jacob, Aziz Naanthaamornphong, Christoper Corley, Jonathan Corley, Brian Eddy, Dustin Heaton, Elizabeth Williams, Amber Krug, Wenhua Hu, and Ahmed Al-Zubidy. Without your great and enjoyable company, it was not possible for me to pass the long journey here. To my friends in Tuscaloosa, Ashfakul Islam, Kazi Zunnurhain, Sufal Biswas, and Mohammad Asadul Hoque, and many others. Thank you

everyone, for those memorable moments, and great support.

Finally, I would like to thank the financial support provided by the Department of Computer Science at UA, as well as the funding support from the National Science Foundation (Grant: 1322276) and the National Security Agency.

CONTENTS

ABSTRACT	ii
DEDICATION	iv
ACKNOWLEDGMENTS	v
LIST OF TABLES	xiii
LIST OF FIGURES	xv
1 INTRODUCTION	1
1.1 Related Work	6
1.2 Research Objective	9
1.2.1 Importance of code reviews	9
1.2.2 Code review process	9
1.2.3 Differences between code review practices in OSS and commercial projects	9
1.2.4 Characteristics of code review social networks	10
1.2.5 Non-technical benefits of code reviews	10
1.2.6 Impact of human factors on code reviews	10
1.3 Organization	10
1.4 Key Findings and Contributions	13
1.5 Outline of the Dissertation	14
2 PEER IMPRESSIONS FORMATION IN OSS ORGANIZATIONS:	
MOTIVATIONAL STUDY	15
2.1 Research Questions and Hypotheses	17
2.1.1 Experiences working with other participants	17
2.1.2 Perceived Expertise	17

2.1.3	Trust	18
2.1.4	Losing Mutual Trust	18
2.1.5	Meeting in Person	18
2.1.6	Belief about Peers' Impressions	19
2.1.7	Judging Peer Productivity	20
2.1.8	Judging a Peer as Easy or Difficult to Work With	20
2.2	Survey Design	20
2.3	Data Analysis	21
2.4	Respondent Demographics	22
2.4.1	OSS Projects Represented	22
2.4.2	Project Role(s) of the Respondents	22
2.4.3	Voluntary vs. Paid-Participation	23
2.4.4	Distribution of Participants across Organizations	24
2.5	Results	24
2.5.1	RQ1: Experiences working with other participants	24
2.5.2	H1: Impact of Perceived Expertise on Peer Interactions	27
2.5.3	H2: Impact of Trust on Peer Interaction	28
2.5.4	RQ2: Losing trust in a peer	29
2.5.5	H3: Effect of Meeting in Person on Peer Impression	30
2.5.6	H4: Accuracy of Peer Impression	32
2.5.7	H5: Effect of Project Contributions on Opinions about Productivity and Competency	33
2.5.8	RQ3: Factors affecting Ease or Difficulty of Working Together	35
2.6	Threats to Validity	37
2.6.1	Internal validity	37
2.6.2	Construct validity	37
2.6.3	External validity	38
2.7	Conclusion	38
2.8	References	39

Appendices	44
2.A Survey questions	44
3 INSIGHTS INTO THE PROCESS ASPECTS AND SOCIAL DYNAMICS OF CONTEMPORARY CODE REVIEW	48
3.1 Background	50
3.1.1 Contemporary Code Review Process	50
3.1.2 Overview of Contemporary Code Review Research	50
3.2 Research Questions	53
3.2.1 Importance of Code Review	53
3.2.2 Code Review Process	53
3.2.3 Impact of Code Review on Peer Impressions	54
3.3 Research Method	55
3.3.1 Survey Design	55
3.3.2 Participant Selection	56
3.3.3 Pilot Tests	57
3.3.4 Data Collection	57
3.3.5 Data Processing and Analysis	58
3.4 Demographics	59
3.4.1 Projects represented	59
3.4.2 Respondent demographics	60
3.5 Results	60
3.5.1 RQ1: Importance of code reviews	61
3.5.2 RQ2: Time spent in code reviews	66
3.5.3 RQ3: Accepting review request	67
3.5.4 RQ4: Characteristics of low quality code	70
3.5.5 RQ5: Assistance to improve low quality code	72
3.5.6 RQ6: Impact of high quality code	76
3.5.7 RQ7: Impact of low quality code	79
3.5.8 RQ8: Effect on peer impressions	82

3.6	Discussion	84
3.6.1	Benefits of Code Review	84
3.6.2	Peer Impression Formation	84
3.6.3	Effects on Future Collaborations	85
3.6.4	Effects of Distributed vs. Co-located Teams	85
3.6.5	Differences Between OSS and Microsoft	86
3.6.6	Effects of Perceived Expertise	87
3.7	Threats to Validity	87
3.7.1	Internal validity	87
3.7.2	Construct validity	88
3.7.3	External validity	88
3.7.4	Conclusion validity	89
3.8	Conclusion	89
3.9	References	90
	Appendices	95
3.A	Survey Questions	95
4	IMPACT OF DEVELOPER REPUTATION ON CODE REVIEW OUTCOMES IN OSS PROJECTS	99
4.1	Background	101
4.1.1	Contemporary Code Review	101
4.1.2	Social Network Analysis	102
4.1.3	Core-Periphery Structure in OSS Communities	103
4.2	Research Hypotheses	103
4.2.1	First Feedback Interval	104
4.2.2	Review Interval	105
4.2.3	Code Acceptance Rate	105
4.2.4	Number of Patchsets per Review Request	105
4.2.5	Number of Review Comments	106

4.2.6	Patchset size	106
4.2.7	Number of Reviewers per Review Request	106
4.3	Core Identification using K-means (CIK)	106
4.3.1	Data Collection and Preparation	107
4.3.2	Building Social Networks	108
4.3.3	Core-Periphery Detection using the CIK Approach	108
4.4	Data Analysis and Results	109
4.4.1	Analysis Approach	109
4.4.2	H1: First Feedback Interval	112
4.4.3	H2: Review Interval	113
4.4.4	H3: Acceptance Rate	115
4.4.5	H4: Number of Patches per Review Request	117
4.4.6	H5: Number of Review Comments	117
4.4.7	H6: Patchset size	119
4.4.8	H7: Number of Reviewers per Review Request	120
4.5	In-depth Analysis	120
4.5.1	Inter/intra group collaboration	122
4.5.2	Longitudinal analysis of reputation	124
4.6	Threats to Validity	125
4.6.1	Internal validity	125
4.6.2	Construct validity	125
4.6.3	External validity	126
4.6.4	Conclusion validity	127
4.7	Conclusion	127
4.8	References	129
5	Overall Conclusion	134
5.1	Key Outcomes	134
5.1.1	<i>RO1: Better understand developers' rationale behind practicing code reviews.</i>	134

5.1.2	<i>RO2: Better understand contemporary code review process.</i>	135
5.1.3	<i>RO3: Study the differences between code review practices in OSS and commercial projects.</i>	135
5.1.4	<i>RO4: Better understand the characteristics of code review social networks.</i>	136
5.1.5	<i>RO5: Identify various non-technical benefits of code reviews.</i>	136
5.1.6	<i>RO6: Identify the human factors that influence code review process and its outcomes.</i>	137
5.2	Key Contributions	137
5.3	Directions for Future Work	138
5.4	Publications	139
5.4.1	Journal Papers	140
5.4.2	Refereed Conference Papers	140
5.4.3	Doctoral Symposium / Workshop Papers	141
5.4.4	Refereed Short Papers	142
	REFERENCES	143
A	APPENDICES	146

LIST OF TABLES

2.1	Respondents' primary OSS project	23
3.1	Coefficient Alpha (Cronbach's α) of the scales	58
3.2	Respondents' primary projects	59
3.3	Demographics of the respondents	60
3.4	Behavioral scale means and effects	84
3.5	Behavioral Scale Questions	98
4.1	Social Network Centrality Measures	110
4.2	Core-Periphery partitioning of the projects	111
4.3	Statistical Results for H1: First interval	112
4.4	Statistical Results for H2: Review interval	114
4.5	Statistical Results for H3: Acceptance rate	115
4.6	Statistical Results for H4: Number of patchsets	117
4.7	Statistical Results for H5: Number of comments	118
4.8	Statistical Results for H6: Number of patchset size	119
4.9	Statistical Results for H7: Number of reviewers	122
4.10	Statistical Results for Group differences	122

LIST OF FIGURES

1.1 Simplified code review workflow	3
1.2 Gerrit code review tool	3
1.3 Example code reviews: 1) style inconsistency, 2) redundant check, and 3) XSS vulnerability	5
1.4 Organization of the Dissertation	12
2.1 Roles of the respondents	23
2.2 OSS participants' experiences working with peers	24
2.3 Experiences working with peers	27
2.4 Details of experiences working with peers	27
2.5 Importance of peer's a) perceived expertise, and b) trustworthiness for interaction	28
2.6 Factors that cause a peer to lose trust	29
2.7 Participants' belief that meeting in person affects their impressions	30
2.8 Opinions regarding meeting in person, paid participants vs. volunteers	31
2.9 Participants' belief that peers have accurate impressions	32
2.10 Importance ratings of the factors to decide a peer as productive or competent	34
2.11 Differences in distribution based on respondents' impressions about peers	36
2.12 Factors influencing how easy it is to work with a peer	36
2.13 Factors influencing the decision to judge a peer difficult to work with	37
3.1 Simplified code review workflow	51
3.2 Importance of code reviews	62
3.3 Hours spent in code reviews vs. Experience	67
3.4 Why the identity of a code author is relevant	68
3.5 Characteristics indicating poor code (Sorted based on the ranks in the OSS survey)	71
3.6 How reviewers assist to fix a poor code	73
3.7 Impact of a high quality code	77
3.8 Impact of a poor quality code	80

3.9	distribution of ratings for the scale items: Perception of expertise	83
4.1	A small network with core-periphery structure (adapted from Borgatti and Everett (2000)). Core nodes are colored using red.	104
4.2	Code Review Social Network Diagram of two OSS Projects	107
4.3	First Feedback Intervals	112
4.4	Experience vs. First feedback interval (Qt project)	113
4.5	Review Intervals	114
4.6	Experience vs. Review interval (Chromium OS)	115
4.7	Acceptance rate	116
4.8	Experience vs. Acceptance Rate (OVirt)	116
4.9	Average number of patchsets per review request	117
4.10	Number of review comments	118
4.11	Patchset size	120
4.12	Experience vs. Code churn (OmapZoom)	121
4.13	Number of reviewers	121
4.14	Differences in first feedback intervals during inter / intra group collaboraions . .	123
4.15	Developers moving from periphery to core while starting as newcomer a) running median first feedback interval (left), b) running median review interval (middle), 3) running average acceptance rate (right)	125

CHAPTER 1

INTRODUCTION

In 1976, Michael Fagan proposed software inspections in which developers subject their code to review by peers or other stakeholders to identify defects (Fagan, 1976). Since then, it has been perceived as an effective quality improvement practice (Wiegers, 2002; Cohen et al., 2006). Even with the benefits offered by software inspections, their relatively high cost and time requirements have reduced the prevalence with which software teams, in general, adopt them (Johnson, 1998).

After observing time loss and difficulties in scheduling formal inspection meetings, Votta Jr (1993) raised the question whether formal meetings are really necessary for code inspections. He suggested that reviewers and authors can correspond using verbal, written, or electronic medium without actually meeting physically. As a support for this observation, a few Open Source Software (OSS) communities (e.g., Apache (Rigby et al., 2008) and Linux (Asundi and Jayant, 2007)) championed the peer code review (i.e., informal software inspection) process using electronic communication (i.e., mailing list). Recently, many mature and successful OSS projects (e.g., Android, Chromium, Eclipse, FreeBSD, Mozilla, OpenStack, OVirt, Qt, and WikiMedia) as well as commercial organizations (e.g., Microsoft, Google, Facebook, and Cisco) (Bacchelli and Bird, 2013) have adopted peer code reviews as an important quality assurance mechanism. Researchers have termed the current peer core review practices as *Contemporary Code Review* (Rigby and Bird, 2013). However, there are marked differences between the *Fagan-style code inspection* and the *contemporary code review* practiced today by many software projects.

1. A Fagan inspection is a heavyweight process requiring synchronous meeting between the participants in multiple phases. On the other hand, contemporary code review practices are asynchronous and light-weight in terms of formalities. Moreover, contemporary code review also does not require synchronous meeting between the participants.

2. Formal inspection does not have any specialized tool support. On the other hand, many specialized tools are available to facilitate contemporary code review process. There is an increasing trend among the projects to use tools (e.g., Gerrit ¹, Phabricator ², Crucible ³, and ReviewBoard ⁴) to support the code review process.
3. Despite being around for more than four decades, the adoption rate of formal inspection was low due to its formal requirements and time constraints. Conversely, contemporary code review has addressed many shortcomings of Fagan inspection and has shown increasing adoption in industry and OSS contexts (Balachandran, 2013; McIntosh et al., 2014; Rigby and Bird, 2013).

Although there are differences among code review tools, most tools support a similar code review workflow. Figure 1.1 provides a simplified overview of the contemporary code review process. A review begins with an author creating a patch-set (i.e. all files added or modified in a single revision) of changes, writing a description of the changes, and submitting those to the code review tool. Most of the code review tools also support selecting reviewer(s) for the code change. Upon creation of a code review request, the review tool notifies the reviewer(s) about the incoming review via email. Code review tools facilitate the review process by highlighting the changes between the revisions in a side-by-side display. For example, figure 1.2 shows an example side-by-side display from Gerrit, one of the most popular open-source code review tools. Both the reviewers and the author can add inline comments highlighting portions of the code. The author addresses the review comments by making changes to the code and then uploading the changes to the review tool to initiate a new iteration of the review. This review cycle repeats until the reviewer(s) approve the changes or the author abandons the change. If the reviewer(s) approve the code, the author checks in the code to the project repository.

Figure 1.3 shows three example issues identified during code reviews. In the first example, the author has used camelCase for the highlighted line (i.e., line 83), although all other lines used underscore. This example is a style inconsistency issue identified by the reviewer. In

¹ <https://code.google.com/p/gerrit/>

² <http://phabricator.org/>

³ <https://www.atlassian.com/software/crucible/overview>

⁴ <https://www.reviewboard.org/>

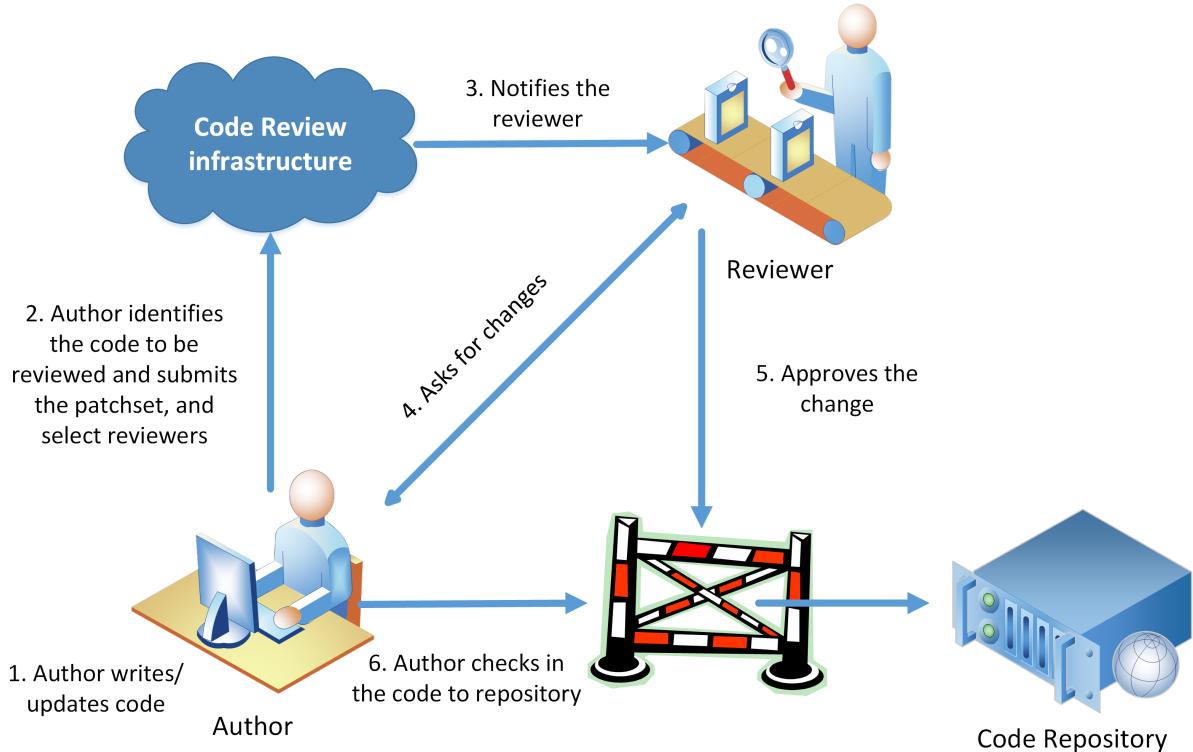


Figure 1.1: Simplified code review workflow

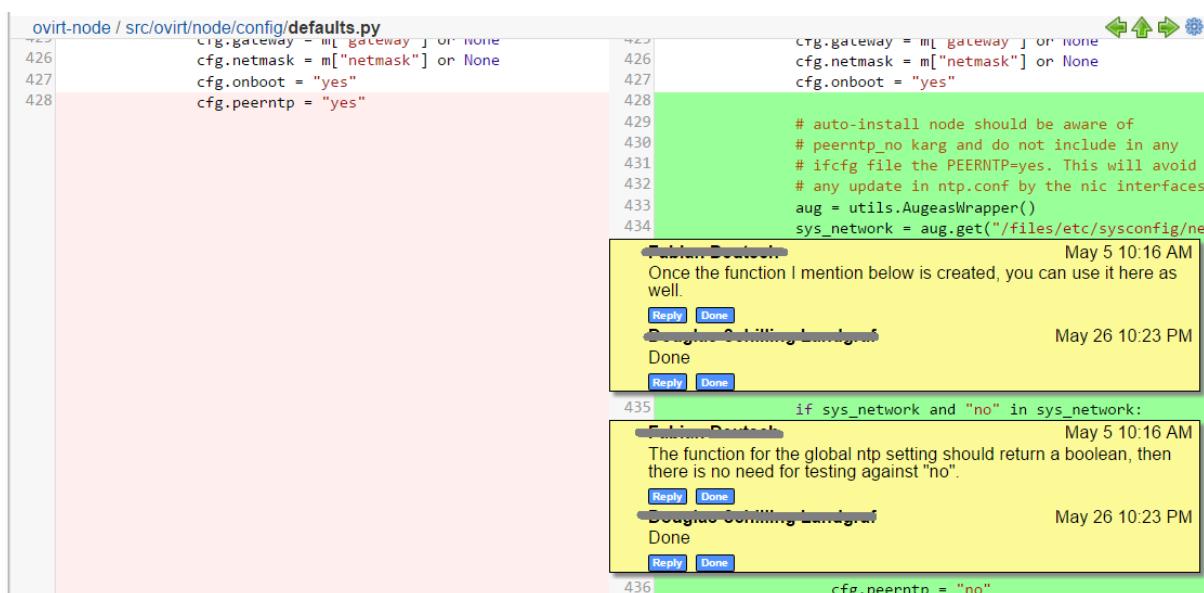


Figure 1.2: Gerrit code review tool

the second example, the author has used a redundant check for the variable ‘noMedia’ inside the inner if block (i.e., line 482). This example is a minor redundant check issue identified by the reviewer. Although a redundant check may not be a defect, it may have performance impact. In the third example, the author has used an unsafe HTML handling method (i.e., line 57). This has the potential to open cross-site scripting attach, which is a critical security vulnerability. These three examples, indicate that code reviews can be useful in identifying a wide range of issues, from style inconsistencies to redundant checks to even critical security flaws. As a result many popular OSS projects as well as commercial organizations have adopted code reviews (Bacchelli and Bird, 2013). Because both OSS and commercial developers spend a significant amount of effort performing code reviews (Bosu and Carver, 2013), improving the effectiveness of the code review process will be beneficial for those organizations. Improving a process or practice requires a thorough understanding of current process, and the factors that influence its outcomes. Therefore, *the primary goal of my dissertation is to better understand contemporary code review practices, its benefits, and factors that influence its outcomes.*

Software development requires collaboration and knowledge sharing between a large number of developers and stakeholders. Therefore, the perception that project participants have of each other’s personal characteristics and abilities influence the communication and collaboration between them, as well as overall project productivity. On the other hand, due to the adoption of distributed software development many participants are distributed over different geographical locations. Due to the rarity of face-to-face interactions, members of distributed teams often have a difficult time forming accurate impressions of their peers, which ultimately hurts team productivity. Prior research suggests that members of distributed teams are task-focused and emphasize socio-technical interactions (i.e., interactions between people using technologies in workplaces) to form impressions about their peers more so than do co-located team members (Guadagno and Cialdini, 2005). In a software project, developers may interact via various media (e.g., reviewing code changes, discussing bug fixes, describing code commits, and posting in the mailing-list). Some of those interactions may facilitate more direct interactions between the developers than the others and have more impact on impressions formation. Therefore, *the secondary goal of my dissertation is to understand the impact of dif-*

David Turner Sep 13, 2010
the functions below operate on an AModem object, so should be named amodem_switch_technology(),
amodem_set_cdma_xxxx

Reply ... **Reply 'Done'**

Emilio Lopez Sep 23, 2010
Done **Example#1: Style inconsistency**

Reply ... **Reply 'Done'**

```
static const char* switchTechnology(AModem modem, AModemTech newtech, int32_t newpreferred);
static int set_cdma_subscription_source(AModem modem, ACdmaSubscriptionSource ss);
static int set_cdma_prl_version(AModem modem, int prlVersion);
```

83
84
85
86

Emilio Lopez Jan 15, 2012
Isn't '&& noMedia' redundant? noMedia can only be true inside the if block**Example#2: Redundant check**

Reply ... **Reply 'Done'**

David Turner Jan 16, 2012
Thanks Emilio! My bad :D

Reply ... **Reply 'Done'**

```
54 >>>     if ( $this->isPermalink ) {
55 >>>         >>     $html .= Linker::link(
56 >>>             >>         >>     SpecialPage::getTitleFor( 'ArticleFeedbackv5', $record[0]->
age_title ),
57 >>>         >>         >>     wfMessage( 'articlefeedbackv5-special-goback' )->text());
```

Catoppo Apr 17, 2012
And this unfixes a security bug that was fixed earlier: this must be ->escaped(), not ->text(), because the second param to link() is HTML.
Example#3: XSS vulnerability

Reply ... **Reply 'Done'**

Tomas Styrbin Apr 17, 2012
Done

Reply ... **Reply 'Done'**

Figure 1.3: Example code reviews: 1) style inconsistency, 2) redundant check, and 3) XSS vulnerability

ferent socio-technical interactions on peer impression formation among software developers.

1.1 Related Work

In recent years, there have been several studies on code review practices. Most of those studies focused on understanding contemporary code review processes in different projects. Rigby and German (2006) were the first researchers to examine the code review process in OSS communities. After exploring code review policies and processes of 11 OSS projects, they observed two types of peer code reviews: 1) pre-commit review (Review Then Commit-RTC), and 2) post-commit review (Commit Then Review - CTR). In a subsequent study of the Apache project, they found that the CTR is faster than the RTC but is not less efficient in bug detection. They concluded Apache code reviews are efficient and effective because of the “early, frequent reviews of small, independent, complete contributions conducted asynchronously by a potentially large, but actually small, group of self-selected experts.” (Rigby et al., 2008)

To characterize the code review practices, (Rigby and German, 2006) also proposed a set of code-review metrics (i.e. acceptance rate, reviewer characteristics, top reviewer vs. top committer, review frequency, number of reviewers per patch, and patch size). Other researchers calculated similar metrics for five OSS projects and concluded that code review practices vary across different OSS projects based on age and culture of the projects (Asundi and Jayant, 2007). However, these findings were contrasted in a later study, which found that despite large differences among five OSS projects and several commercial projects, their code review metrics were largely similar (Rigby and Bird, 2013). While those studies provide several key insights to better understand of contemporary code review, no study has explored several key areas (e.g., developers perception of the importance of code reviews, social interaction during code reviews, and best strategies to assist poor quality code). A better understanding of these unexplored areas can help project managers decide upon the usefulness of code reviews and help developers improve their review process. Therefore, this dissertation aims to provide a better understanding of these unexplored areas.

In terms of understanding the code review process, earlier studies focused on understanding decisions mechanism during code reviews. In a study of five OSS projects that used broadcast

based review (i.e. sending patches to a mailing list for review), Rigby and Storey (2011) found that reviewers decide to review patches based on their own interests and past collaborations, therefore code changes failing to generate interests among the core developers tend to be ignored. Core developers of OSS projects are aware that too many opinions on a patch leads to unrelated and unproductive discussions, therefore they try to avoid longer threads to ensure efficient broadcast based code review. While the work of Rigby and Storey (2011) provided an understanding of developers decisions on review participation during mailing-list based code reviews, those decisions during tool-based code reviews remain unexplored. A better understanding of reviewers' criteria for accepting code review requests may help practitioners in selecting appropriate reviewers for their code. Moreover, an understanding of how reviewers evaluate the quality of code changes may help code authors write code that is easier to review. This dissertation focuses on providing insight into these two criteria.

After seeing the successful adoption of code review practices by OSS projects, many commercial organizations have recently adopted peer code review practices (Sutherland and Venolia, 2009; Rigby et al., 2012; Bacchelli and Bird, 2013; Balachandran, 2013). Contrary to OSS projects, code review participants at Microsoft use both synchronous and asynchronous communication mediums for code reviews and consider communications during code reviews essential to understand code changes and design rationale later. Microsoft developers expressed a need to retain code review communications for later information needs (Sutherland and Venolia, 2009). Another study at Microsoft found that although finding defects is a primary motivation for code reviews, other benefits (e.g. knowledge dissemination, team awareness, and identifying better solutions) may be more important. The major challenge is understanding the code changes (Bacchelli and Bird, 2013). While these studies characterized the code reviews in commercial projects, only one study (Rigby and Bird, 2013), which focused on quantitative aspects of code reviews, has compared and contrasted between the code review practices of OSS and commercial projects. Since developers' motivations and project governance differ between OSS and commercial organizations, code review collaborations may also differ between OSS and commercial projects. This dissertation aims to explore those differences.

While most of the earlier exploratory studies focused on understanding the code review

practices, a few of the recent studies have focused on understanding the impact of different factors on code reviews. Code review characteristics, such as review size, component, priority, organization, reviewer characteristics, and author experience significantly influence on both review completion time and outcome (Baysal et al., 2013). Moreover, a reviewer's prior experience in changing or reviewing the artifact under review and project experience increases the likelihood of providing useful feedback (Bosu et al., 2015). While these studies focused on technical human factors and characteristics of the code changes, no studies have focused on the non-technical human factors (i.e., author's reputation, and relationship between an author and a reviewer). Because code review facilitates direct collaboration between people, a better understanding of the impacts of various human factors is crucial to improve the code review process. Therefore, this dissertation also focuses on identifying the impact of non-technical human factors on code review outcomes.

A few recent studies have investigated various technical benefits of code reviews. Although the primary goal of code reviews is defect detection, because three-fourths of the review comments are related to maintainability issues code review may be more beneficial for projects which require highly maintainable code (Beller et al., 2014). Code reviews have significant impact on software quality. A recent study found that both low code review coverage (i.e., the proportion of changes that have been reviewed) and low review participation (i.e., the number of reviewers) often increase the likelihood of post-release defects (McIntosh et al., 2014). While these studies focused on the technical benefits of code review, only one study (Bacchelli and Bird, 2013) has explored the non-technical benefits of code reviews. The evidence about the non-technical benefits (i.e., impressions formation, knowledge sharing, and mentoring) has been mostly anecdotal. Empirical evidence regarding various benefits of code reviews can encourage project managers to adopt code reviews for their projects. Therefore, this dissertation aims to bridge this research gap by providing evidence regarding the non-technical benefits of code reviews.

1.2 Research Objective

The primary objective of this dissertation is to better understand contemporary code review practices, its benefits, and factors that influence its outcomes. I approach this high-level objective into six specific objectives. Following subsections describe the specific objectives of this dissertation.

1.2.1 Importance of code reviews

Although, the primary goal of code review is defect detection, recent studies indicate that only around one-fourth code review comments find functional defects in the code (Bosu et al., 2015; Beller et al., 2014). Therefore, the first objective of this dissertation is to:

RO1: *Better understand developers' rationale behind practicing code reviews.*

1.2.2 Code review process

Prior studies have contributed better understanding of types of code review practices (Rigby and German, 2006), review selection during broadcast based code reviews (Rigby and Storey, 2011), and code review durations (Rigby and Bird, 2013). However, several areas of contemporary code reviews require better understandings (e.g., how developers decide to accept/reject incoming review requests, how developers judge the quality of a code change, and how developers assist poor code changes). Therefore, the second objective of this dissertation is to:

RO2: *Better understand contemporary code review process.*

1.2.3 Differences between code review practices in OSS and commercial projects

Most of the prior studies focused on a small number of projects with similar characteristics. Prior studies suggest differences in motivations between OSS and commercial software developers (Raymond, 1998). Again the governance of OSS and commercial projects are different. Therefore, the code review practices in OSS projects may differ from the practices in commercial projects. Although Rigby and Bird (2013) compared between code reviews in OSS and commercial projects, they only focused on a small number of quantitative metrics. This dissertation aims to:

RO3: *Study the differences between code review practices in OSS and commercial projects.*

1.2.4 Characteristics of code review social networks

When a developer submits a code change for review, he may have several options to select reviewers for that change. For example, he may choose a developer 1) who works in his team, 2) who he considers as an expert, 3) who had provided useful reviews before, or 4) who often asks for review. The social interaction network based on code review interactions for a project can provide us better understandings of how developers collaborate during code reviews. Therefore, the next objective of this dissertation is to:

RO4: *Better understand the characteristics of code review social networks.*

1.2.5 Non-technical benefits of code reviews

Although, there has been a growing interest in the software engineering research community about peer code review in the last few years, there are many areas of contemporary code review practices that warrant additional research. For example, although there is empirical evidence that code review improves software quality, the evidence about the other benefits of code review (e.g., sharing knowledge, sharing expertise, sharing development techniques, and most importantly forming peer impressions) has been mostly anecdotal. Therefore, this dissertation aims to:

RO5: *Identify various non-technical benefits of code reviews.*

1.2.6 Impact of human factors on code reviews

Because, code review requires collaboration between two human peers, there should be many human factors (e.g., relation between the peers, and reputation) affecting code review process. However, most of the prior studies did not address those human factors. Therefore, this dissertation aims to:

RO6: *Identify the human factors that influence code review process and its outcomes.*

1.3 Organization

This dissertation consists of three journal articles on peer impression formation and contemporary code review. Since software development requires collaborations between developers,

the perception that project participants have of each other's personal characteristics and abilities influence the communication and collaboration between them, as well as project outcomes. To understand peer impressions formations and its influences in software development organizations, the articles of this dissertation focus on: 1) how a developer forms impressions about his peers, 2) better understanding a software development practice (i.e., code review) that has favorable characteristics to support impressions formation, and 3) how a developer's impressions about his peers influence his future collaborations, as well as the outcomes of his contributions. Figure 1.4 shows organization of the dissertation based on the three articles. The first article is published and the remaining two articles are ready for submission.

The first article describes a survey of developers from popular OSS projects to understand peer impression formation among distributed team members (Bosu et al., 2014). The goals of that study were to understand:

- how different forms of peer impressions develop in OSS communities,
- the factors that affect the impression formation process,
- how peer impressions evolve, and
- the opinions of OSS participants about those peer impressions.

The results of the survey indicated that coding-related factors have the most influence on impression formation among OSS developers. Therefore, I hypothesized that interactions in which participants can judge a peer's code or a peer's creativity would be crucial for peer impressions formation. Among various software development practices, code review facilitates direct interactions to judge a peer's code. Therefore, I hypothesized that contemporary code reviews would have favorable characteristics to support mutual impressions formation. Based on this hypothesis, the goal of this dissertation is to better understand contemporary code review process and its associated benefits.

To address this goal, the second article describes the results of a survey of code review participants from OSS projects and from Microsoft. First, I surveyed the developers from 36 popular OSS projects about 1) the code review process in their projects, 2) their expectations from code review, and 3) how code review impacts impressions about their peers. The results

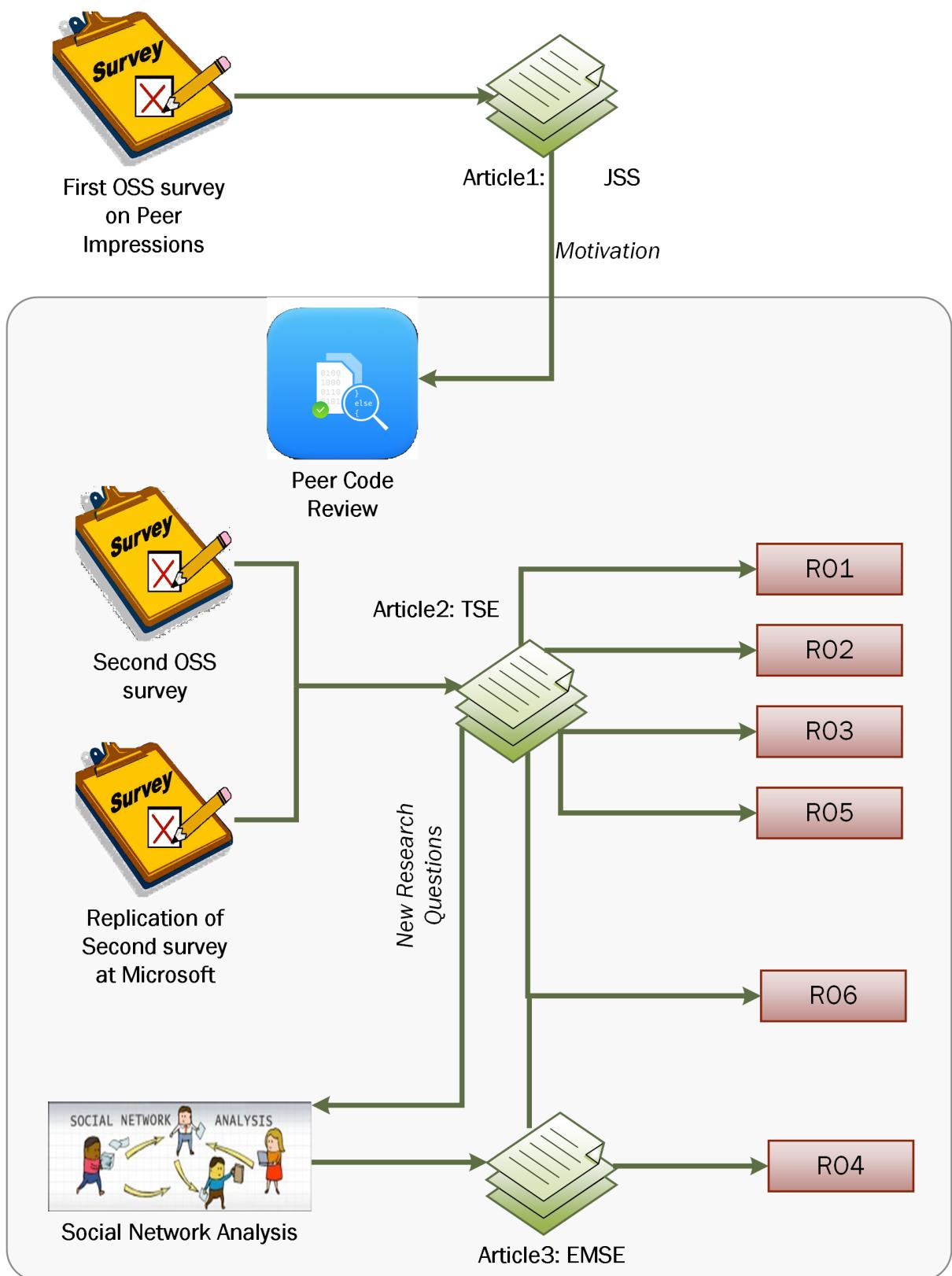


Figure 1.4: Organization of the Dissertation

of the surveys provide insights on five research objectives (i.e., RO1, RO2, RO3, RO5, and RO6) of this dissertation. Since developers' motivations and project governance differ between OSS and commercial organizations, I hypothesized that there would be differences between the code review process of OSS and commercial projects. To study this hypothesis, I replicated the survey at a large-scale commercial software organization (i.e., Microsoft). The results of the code review surveys indicated that code reviews help developers building perceptions of each other's abilities and personal characteristics.

To make the results from the surveys useful, it is important to understand how a developer's impressions about his/her peers may impact future collaborations, and more importantly the outcomes of his/his contributions. To investigate this research question, I studied eight OSS projects using software repository mining and social network analysis techniques. The third article describes the results of this study providing empirical evidence regarding the impact of a developer's reputation on the outcomes of his/her contributions. The results of the study provide insight for RO4 and RO6 of this dissertation.

1.4 Key Findings and Contributions

The results of this dissertation provide key insights about contemporary code reviews and impact of socio-technical interactions on peer impression formation. When forming impressions about a peer, developers emphasize on the quality of that peer's contributions to the project. As a result, those types of interactions where participants can have the opportunity to judge a peer's code or creativity (e.g., code review) are very influential in peer impression formation. Code review is an important software engineering practice that not only identifies defects but also helps knowledge sharing, facilitates impression formation, and improves project maintainability. Code changes that are easy to understand and require minimum review effort are judged highly by the reviewers and increase the reputation of the author. Reputation not only impacts the code review process by facilitating quicker feedback and higher acceptance rate but also improves future collaborations among developers.

The key contributions of this dissertation are:

- Empirical insights into the peer impression formations in OSS organizations.

- Better understanding of developers' perceived benefits of contemporary code reviews.
- Empirical evidence regarding various technical and non-technical benefits of code reviews.
- Better understanding of why and how developers collaborate during code reviews.
- Empirical evidence regarding the effect of an OSS developer's reputation on the outcome of his/her code review requests.

1.5 Outline of the Dissertation

This dissertation is divided into five chapters. Chapter 2 presents the results of the study on peer impressions formation in OSS projects. Chapter 3 compares the results of the surveys on peer code review practices in OSS projects and Microsoft. Chapter 4 presents the results of the social network analysis study to find out the impact of developer reputation on code review outcome. Finally, chapter 5 summarizes the findings of this dissertation and provides directions for future research.

CHAPTER 2

PEER IMPRESSIONS FORMATION IN OSS ORGANIZATIONS: MOTIVATIONAL STUDY

Many expert developers devote a significant amount of effort to Open Source Software (OSS) projects. Because many of those participants are not directly compensated, their participation must be motivated by other factors. Previous empirical research found that OSS participants are motivated by the prospects of enhancing their reputation and by being identified with a particular OSS community. According to Raymond (1999): “The ‘utility function’ Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers.” Reputation among one’s peers is the only available measure of competitive success (Raymond, 1998) and the main source of power (Evans and Wolf, 2005) in OSS communities. Furthermore, the participants’ desire to maintain a good reputation among their peers is a major motivation for voluntarily devoting effort to an OSS project (Gutwin et al., 2004). The level of dedication a participant has for the OSS project is strongly related to peer recognition (Xu and Jones, 2010). Therefore, gaining and maintaining reputation is a key factor in keeping an OSS project on track (Markus et al., 2000). When a participant is well-recognized within the OSS community, his or her peers regard the participant’s project related opinions more carefully (Gacek and Arief, 2004).

Because gaining peer recognition is a major motivation for OSS participants and it influences OSS projects greatly, it is important to understand the peer recognition process within OSS communities. We define **peer recognition** as: *the acknowledgement of a person’s merits or status by his or her peers.* Before a person can acknowledge the merits or status of a peer s/he must be aware of those merits or status. Therefore, it is important to understand how peers form opinions of each other in OSS communities. There is a large body of research on peer recognition in the Psychology literature, where researchers use the term “impression formation” to describe the same concept. According to Kenny (1994), *peer impression* is “the

judgements that a person, called the perceiver, makes about another person, called the target, where the target is a real person.” The formation of interpersonal impression primarily depends upon how well the perceiver is acquainted with the target and upon the personality traits of those two individuals. Similarly, Moore (2007) defined impression formation as: “the process by which individuals perceive, organize, and ultimately integrate information to form unified and coherent situated impressions of others.”

Generally, members of OSS communities are geographically distributed, rarely or never meet face-to-face (FTF), and collaborate using text-based tools over the Internet, i.e., Computer-Mediated-Communication (CMC) (Guadagno and Cialdini, 2005; Jarvenpaa and Leidner, 1998). Research has shown a marked difference between FTF and CMC regardless of the purpose. Specifically, McKenna and Bargh (2000) propose four domains in which social interaction via CMC differs from other more conventional interaction media: relative anonymity, reduced importance of physical appearance, attenuation of physical distance, and greater control over the time and pace of interactions. Due to those differences, the impression formation process between OSS participants is different than the impression formation process between co-located project participants. However, there is a lack of knowledge about the impression formation between the participants of OSS projects (Marlow et al., 2013).

To better understand the formation and evolution of peer impressions in distributed OSS teams, we surveyed a broad spectrum of OSS participants to discover: 1) how different forms of peer impressions develop in OSS communities, 2) the factors that affect the impression formation process, 3) how peer impressions evolve, and 4) the opinions of OSS participants about those peer impressions. In this study, we primarily focused on five dimensions of peer impressions: 1) productivity, 2) competency, 3) easy or difficult to work with, 4) perceived expertise, and 5) trustworthiness.

The remainder of the chapter is organized as follows. Section 2.1 presents the research questions and hypotheses for the survey. Section 2.2 describes the survey design. Section 2.3 explains the data analysis process. Section 2.4 discusses the respondent demographics. Section 2.5 presents the results relative to the research questions and hypotheses. Section 2.6 explains the threats to validity of the survey. Finally, Section 2.7 concludes the chapter and

describes planned extensions for a journal submission.

2.1 Research Questions and Hypotheses

Our study of the literature on reputation, communication and collaboration in OSS communities, identified eight important topics that can provide insight into the peer impression process in OSS communities. For five of those topics, the literature presented enough evidence to pose definite hypotheses. For the other three topics, we simply pose research questions, which may lead to hypotheses for future study. This section provides a brief discussion of the literature to motivate each of the five hypotheses and three research questions.

2.1.1 Experiences working with other participants

The ‘craftsmanship model’ states that the pure joy of developing software is a major motivation for OSS participants (Raymond, 1998). Studies have identified the most important reasons why developers contribute to OSS projects to be: enjoyment, learning benefits (Hars and Ou, 2002; Lakhani and Wolf, 2005), and positive experiences from participation (Xu and Jones, 2010). Conversely, if an OSS participant’s experiences are continually negative, he or she will eventually leave the project (Von Krogh et al., 2003). Therefore, we expect that participants in successful OSS projects will have positive experiences. Even so, it is likely that some OSS participants will have negative experiences. To better understand impression formation, it is important to understand the OSS participants’ experiences working with their peers and what factors affect those experiences. Therefore, we pose the following research question:

RQ1: What positive and negative experiences do OSS participants have while working with their peers?

2.1.2 Perceived Expertise

An OSS community member gains reputation primarily based upon his or her consistent high quality contributions (Raymond, 1998). Most OSS activities are highly knowledge-intensive and require a certain level of expertise (Von Krogh et al., 2003). Therefore, an OSS participant displays expertise through his or her contributions to the project. The impression that people have about another person’s expertise affects whether they trust that person’s opin-

ions (Moorman et al., 1993). This finding is true both on and off line (Cialdini, 2009; Guadagno and Cialdini, 2005). The interaction between perceived expertise and interpersonal interaction leads to the following hypothesis:

H1: An OSS participant considers his or her impression of a peer's expertise an important factor affecting their interactions with that peer.

2.1.3 Trust

Psychological research has demonstrated the importance of trust in establishing online relationships (Green, 2007). Similarly, research on team performance suggests that a virtual team needs a solid foundation of mutual trust to enable effective collaboration (Jarvenpaa and Leidner, 1998; Holton, 2001; Peters and Manz, 2007). Virtual teams cannot be effective without trust, because individual members are not willing to take the risk that a team member will act in his or her own self-interest, rather than the interest of the team (Zand, 1972). Because OSS teams are a prime example of virtual, online communities, we can hypothesize the following:

H2: An OSS participant considers his or her level of trust of a peer important when interacting with that peer.

2.1.4 Losing Mutual Trust

OSS participants are quite diverse relative to age, race, nationality, and educational background (Ghosh et al., 2002; Lakhani and Wolf, 2005). The participants also have diverse skills and interests. The diversity often causes conflicts (Jensen and Scacchi, 2005), which may result in lost trust. Again, one participant may be very enthusiastic but not as competent as another participant. Hence, his/her repeated failures may cause the project owners to lose confidence in him/her. There may be other reasons that OSS participants lose mutual trust. Because the literature did not provide enough evidence to hypothesize the most important factors, this research question seeks to identify those factors.

RQ2: Which factors influence OSS participants to lose trust in their peers?

2.1.5 Meeting in Person

Most OSS community members are geographically distributed, rarely or never meet in person, and coordinate primarily via text-based communication tools (e.g., mailing list, Internet

Relay Chat (IRC), repositories, and wikis). Because those communication tools cannot capture facial expressions and body language, it may be difficult to understand and interpret the tone of the communication (Peters and Manz, 2007). In addition, research has shown that virtual teams who use FTF meetings for team building and solving complex issues were more effective than teams that did not use FTF meetings (Maznevski and Chudoba, 2000).

OSS participants often meet each other at conferences (e.g., ApacheCon, EclipseCon, PyCon, and MySQL AB conference). Crowston et al. (2007) interviewed OSS participants at five OSS conferences to understand the impact of FTF meetings. The results indicated that meeting in person helps build social ties among developers, which in turn facilitates better interactions. More specifically, one interviewee mentioned that s/he felt more connected to other participants after meeting with them the first time. Another interviewee mentioned that s/he was more comfortable sending other participants email after meeting them (Crowston et al., 2007). These results are similar to results reported in the psychology literature that social cues (such as a little biographical information about a virtual colleague) facilitate the formation of positive impressions (Tanis and Postmes, 2003). The apparent difference between people who interact FTF compared with those who do not leads us to pose the following hypothesis:

H3: When OSS participants meet in person, their impressions of each other improve.

2.1.6 Belief about Peers' Impressions

Participation in OSS projects is highly visible because most of the activities occur via the Internet. OSS repositories and mailing list archives are open for public viewing. This visibility allows members of the community to monitor the performance and behavior of each community member. Members are generally aware of which members do good work and which members violate the community norms (Markus et al., 2000). This transparency should allow community members to form an accurate opinion of their peers' abilities. To maintain awareness between project members, OSS projects use mailing lists, IRC channels, and commit logs (Gutwin et al., 2004). OSS project hosting sites (i.e., GitHub, LaunchPad, and SourceForge) provide participants with dashboards and activity feeds to help them track project plans and other members' contributions (Treude and Storey, 2010). Thus, we can hypothesize that:

H4: OSS participants believe that their peers have an accurate impression of their abilities.

2.1.7 Judging Peer Productivity

Raymond (1998) states that OSS communities have a ‘gift culture’ by saying that “... social status is determined not by what you control but by what you give away.” An OSS community member’s contributions determine his or her identification and reputation in the community (Roberts et al., 2006). Moreover, due to the virtual nature of OSS collaboration, the differences in online vs. FTF communication (McKenna and Bargh, 2000), and the ease with which contributions can be monitored, we believe that project contributions are the most reasonable way to judge a peer. To test this belief, we hypothesize:

H5: OSS participants consider contributions of a peer to the project as the most important factor when evaluating the productivity or competency of that peer.

2.1.8 Judging a Peer as Easy or Difficult to Work With

Multiple factors can affect whether someone believes working with a particular peer is easy or difficult. Factors related to individual characteristics may include: creativity, responsibility, honesty, and use of negative words (e.g., swear words, insults, or racial remarks). Because virtual community members are very task focused (Guadagno and Cialdini, 2002), an OSS participant may also judge a peer based upon the quality of his or her work. For example, in OSS communities, a participant may judge a peer by his or her code. The literature provided no conclusions about which particular factor(s) might be more important for judging whether working with a peer would be easy or difficult. Therefore, to identify these factors, we posed the following question:

RQ3: What factors affect whether working with a peer on an OSS project will be easy or difficult?

2.2 Survey Design

To investigate the hypotheses and questions posed in Section 2.1, we designed an online survey about communication, collaboration, peer evaluation and conflict resolution between

OSS participants. Using our varied expertise (i.e., Software Engineering and Psychology), we worked together to develop a set of 20 questions, of which eight used a 5-point scale, four used multiple-choice answers, and the remaining eight used open-ended answers. For the multiple choice and 5-point scale questions, we drew the answer choices from the OSS literature (Gallivan, 2001; Hertel et al., 2003; Raymond, 1998; Roberts et al., 2006; Stewart and Gosain, 2006; Xu and Jones, 2010), our own experiences, discussions in OSS community forums and examination of OSS mailing lists. This chapter describes the results of the 15 questions that most directly relate to our hypotheses and questions. Appendix A provides a copy of the survey.

To distribute the survey, we identified 50 successful OSS communities (i.e., at least 50,000 downloads) that had active development communities and used development mailing lists for collaboration. Mailing lists are a critical communication tool for OSS communities (e.g., python-dev for the Python project). Because each active developer subscribes to the development mailing list, these mailing lists reach the appropriate survey population. To ensure that our survey request email was distributed to the list, we contacted the list administrators to obtain access/instructions for sending the survey request. In the survey request email, we specified that only recipients who were actively participating in OSS projects should consider responding to the survey. We subscribed to the mailing lists to verify that the survey request was sent out. Our survey request was posted to 48 of the 50 mailing lists during the second week of June 2011. We sent a reminder email to each list one week after the first email. When the rate of responses slowed, approximately two weeks later, we closed the survey. Overall, 115 developers responded.

2.3 Data Analysis

The survey produced three types of data. For the multiple choice questions (nominal data) and the 5-point scale questions (ordinal data), we used the non-parametric Chi-square and Mann-Whitney U tests, respectively. Both tests are robust and do not require normally-distributed data. We used the standard alpha value of .05 for judging significance. For the open-ended questions, we followed a systematic qualitative data analysis process to generate

nominal data that could be analyzed with the Chi-square test. Each author had his/her own role in data analysis. First, Bassett extracted the general theme associated with each response. Next, Carver and Hochstein, with Software Engineering expertise, and Guadagno and McCallum, with Psychology expertise, analyzed these themes to develop an agreed-upon coding scheme for each question.

Using these coding schemes, Bosu and Bassett worked independently to code the responses. After coding, they examined their results to identify any discrepancies. They discussed those discrepancies and were able to resolve most of them. For the small number of discrepancies they could not resolve, Carver provided the tie-breaking vote after listening to the competing rationales.

2.4 Respondent Demographics

This section characterizes the sample based on responses to five questions to help readers properly interpret the applicability of the results.

2.4.1 OSS Projects Represented

The respondents indicated their primary OSS project. Although we sent the survey to the mailing lists of 48 OSS projects, the respondents listed 67 unique projects as their primary projects (Table 2.1). The reason 67 projects were represented in our sample could be because 74% of the respondents indicated that they worked on multiple OSS projects. A participant who is a minor contributor on one of the 48 projects solicited could also be a major contributor to another OSS project. In addition, OSS participants may subscribe to the email lists of other popular OSS projects simply to stay informed about the latest development activities within that community. The survey instructed the respondents to answer the remaining questions based on their experiences on their primary project.

2.4.2 Project Role(s) of the Respondents

Figure 2.1 shows the percentage of respondents who perform various roles (note that a respondent can perform multiple roles and in fact 90% of the respondents reported multiple roles). Most respondents (80%) perform some type of maintenance activities (e.g., fixing bugs,

Table 2.1: Respondents' primary OSS project

Project	# Resp.	Project	# Resp.	Project	# Resp.	Project	# Resp.	Project	# Resp.
Drupal	9	FreeBSD	7	Debian	6	OpenStack	5	Python	4
Adium	3	Chromium	3	Jenkins	3	Macports	3	MediaWiki	3
PHP	3	Asterisk	2	BugZilla	2	Eclipse Mylyn	2	Fedora	2
GNU Emacs	2	PostGreSQL	2	Squid HTTP	2	Subversion	2	Ubuntu	2
AlphaX	1	Apache Commons	1	Apache httpd	1	Apache Tomcat	1	argoUML	1
arowboat.org	1	Bitweaver	1	Bro IDS	1	CCNx	1	Devnull	1
DTC	1	Eclipse SDK	1	Escher CMS	1	Fink	1	FluidSynth	1
Freeciv	1	FusionForge	1	GlassFish	1	GNS3	1	GNU Guile	1
Mandriva	1	Maxilla	1	ModeShape	1	Mozilla	1	NetBSD	1
One Click Orgs	1	OpenBIOS	1	OSM Potlatch	1	PSPP	1	R-Project	1
Hibernate	1	HylaFAX	1	Innodb	1	Jboss AS	1	kiwix	1
Mactex	1	Mahout	1	Rakudo	1	RISC gcc	1	Robot Framework	1
sbuild	1	Subversion-Edge	1	TortoiseHg	1	tunnelblick	1	VirtualBox	1
weboarf	1	Wordpress	1						

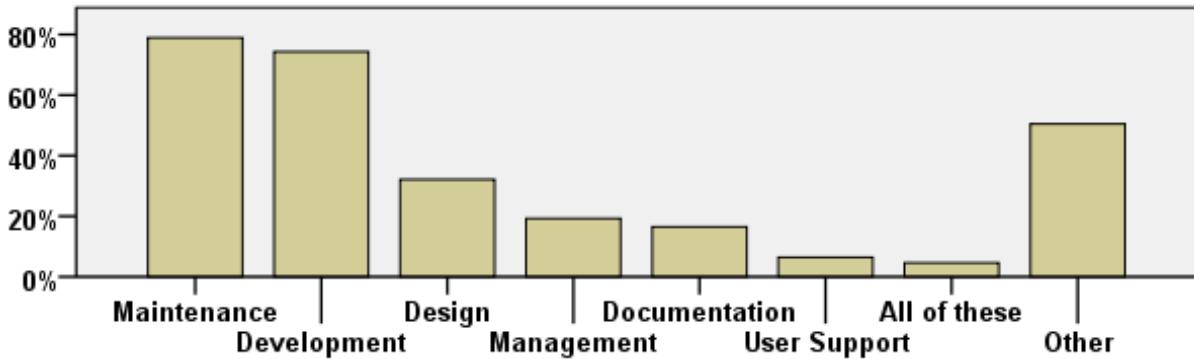


Figure 2.1: Roles of the respondents

maintain infrastructure, package maintenance, bug triage, or general maintenance). About 75% of the respondents perform development activities (e.g., adding new features, committing codes or module integration). About 50% of the respondents performed various other roles (e.g., reporting bugs, participating in discussions, consultancy, system administration, and conducting webcasts).

2.4.3 Voluntary vs. Paid-Participation

Many popular OSS projects are sponsored by commercial organizations (e.g., Google sponsors Chromium and Android). Some employees of these sponsoring companies contribute to the OSS project as part of their job. We call these participants *paid participants*. We call the participants who are not paid to contribute *volunteer participants*. Because paid participants may have different motivating factors than volunteer participants, we expect their behavior may differ. Interestingly, the sample was comprised of approximately the same number of each type of participant (52% volunteer participants and 48% paid). Using this even distribution we

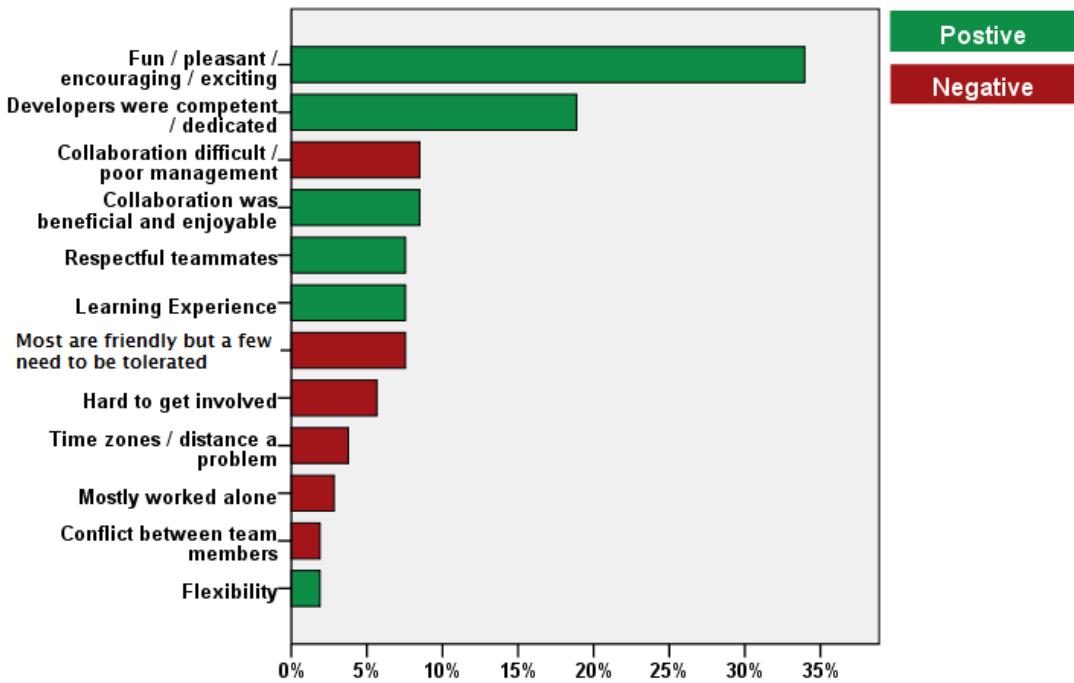


Figure 2.2: OSS participants' experiences working with peers

analyze any differences in responses between the two participant types in Section 2.5.

2.4.4 Distribution of Participants across Organizations

Most respondents (71%) indicated that their projects were highly distributed, i.e., contained contributors from more than five organizations. Conversely, only 13% of the respondents indicated that the projects were co-located, i.e. all participants came from the same organization. The remainder of the respondents (16%) indicated that their projects were distributed across two to five organizations.

2.5 Results

This section describes the analysis for each of the five hypotheses and three research question described in Section 2.1. In addition to analyzing each hypothesis or question in isolation, we also analyzed whether being a paid or volunteer participant affected the results. Where the effect was present, we report the results.

2.5.1 RQ1: Experiences working with other participants

Using the qualitative analysis approach described in Section 2.3, we split the responses into: *Positive Experiences* (65 responses, 61%) and *Negative Experiences* (41 responses, 39%). We

then further divided the responses into sub-groups as shown in Figure 2.2 (note that the total is greater than 100% because each respondent may have provided multiple answers). Overall, the majority of respondents had positive experiences working with their peers, as evidenced by the fact that the two most common codes are positive ones, and all but one of the negative codes appear near the bottom of the distribution. The respondents found their peers to be knowledgeable, respectful, and fun to work with. One respondent commented that:

“There’s folks that are really good at what they do and are a lot of fun to work with.”

Other respondents found collaboration with other participants to be an enjoyable and valuable learning experience. As one respondent put it:

“Most developers have a quite different set of skills and technological background; this also allows me to learn new things as I contribute to the project.”

It is important to examine the negative experiences in more detail to help OSS communities take effective measures to eliminate or reduce the negative experiences. The most common reason for a negative experience was difficulties in collaboration and poor management, typically manifested as difficulty receiving a timely response (9%). Some respondents were frustrated by the bureaucratic decision-making process that made it difficult to obtain a timely and favorable decision about their proposed development directions. Some respondents (about 8%) had good experiences with the majority of the community but had problems getting along with a few people. For example, one respondent stated:

“Generally, the other developers have been attentive to the many problems that I have brought to their notice though there has been one developer who has been over-sensitive to criticism and at times somewhat non-cooperative and aggressive.”

Some participants may even leave a project due to the bad behavior of a peer. As another respondent noted:

“I was very happy working in the group for many years but decided to leave due to the addition of one new developer who was argumentative, short and generally

nasty to users who asked rather basic or dumb questions. I didn't see why I should spend my free time working on a project with such [a rude person]"

Some respondents (about 8%) found it very difficult to get involved with the community due to: lack of mentors, delayed response from project leaders, and a steep learning curve. For example, one respondent said,

"The process of getting involved is a bit tricky - they're very open folks, but getting into somewhere I can assist is difficult because of an incredibly steep learning curve."

As many OSS projects have participants separated by thousands of miles and multiple time zones, some respondents (about 4%) mentioned that time-zone difference and distance were a hindrance to collaborating. If there is a large difference between two participants' time-zones, it is difficult to collaborate synchronously, which can lead to slower resolution of problems. Large geographical distances may result in peers never meeting each other FTF to observe physical cues, which can lead to misunderstood emails and other textual communication. A few respondents (about 2%) also observed conflicts between OSS participants resulting from differences in opinions and clashes of personalities. Although the respondents reported some negative experiences, the overall tones of the responses indicated that the respondents found working with peers to be a positive experience. The following response provides a good summary of the overall experiences of the respondents:

"We're from a wide variety of backgrounds and professions, and therefore bring different perspectives to the project's issues. This can be both good and bad, but usually the good outweighs the bad."

Figure 2.3 shows that the ratio of positive to negative experiences working with peers was significantly dependent upon whether the respondent was a paid or volunteer participant ($\chi^2_{1,106} = 3.559, p = .05$). In addition, Figure 2.4 shows that specific types of experiences were reported with different frequencies by the two types of participants. Some experiences were more common among paid participants while others were more common among volunteer par-

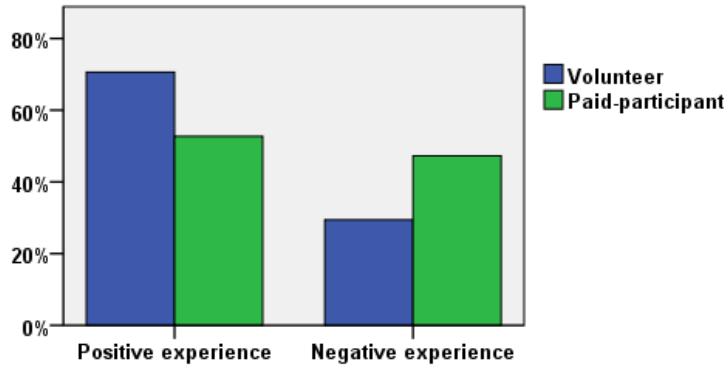


Figure 2.3: Experiences working with peers

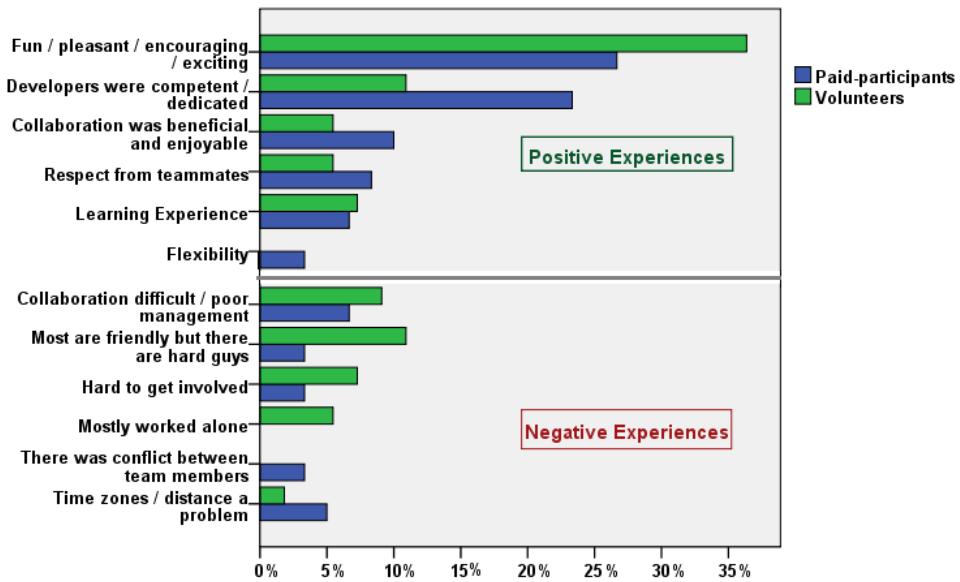


Figure 2.4: Details of experiences working with peers

ticipants. Of those experiences, the only one that was significant was the frequency with which participants found their peers to be competent or dedicated ($\chi^2_{1,106} = 4.771, p = .029$).

2.5.2 H1: Impact of Perceived Expertise on Peer Interactions

Figure 2.5 shows that 56% of the respondents considered their impression of the level of expertise of a peer as a very important or extremely important factor during interaction with that peer. The median and the mode of the ratings were “4 - Very Important.” This positively skewed distribution is significantly different from uniform ($\chi^2_{4,112} = 70.468, p < .001$). This result supports Hypothesis 1. This result is also consistent with previous literature on virtual teams. Potter et al. (2000) highlighted the need for expertise as the primary reason that virtual collaborations form. Peters and Manz (2007) and Kayworth and Leidner (2000) indicated that expertise is an important factor for virtual team performance. Because OSS communities

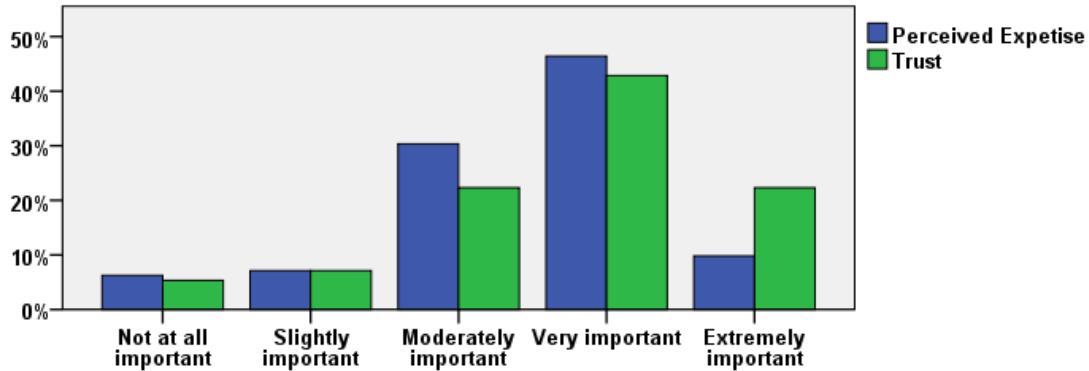


Figure 2.5: Importance of peer's a) perceived expertise, and b) trustworthiness for interaction

are virtual teams (Gallivan, 2001), perception of peer expertise should be an important factor affecting collaboration.

2.5.3 H2: Impact of Trust on Peer Interaction

Figure 2.5 shows that 63% of the respondents considered their level of trust in a peer to be either very important or extremely important during interaction with that peer. The mode and median of the ratings was “4 - Very Important.” This positively skewed distribution is significantly different from uniform ($\chi^2_{4,112} = 54.125, p < .001$). This result supports Hypothesis 2.

This result may be related to the importance of mutual trust in OSS communities found in the literature. Collaborative development efforts, such as those in OSS communities, cannot be effective without mutual trust. Because contributions to the OSS project from different participants have to work together without conflict, the value of one participant’s contribution depends, in part, upon the efforts and contributions of the others (Stewart and Gosain, 2006). A developer also needs to trust that the project administrators will give appropriate credit for his or her contribution. Low levels of trust can result in higher developer turnover (Von Krogh et al., 2003). Other researchers have found that mutual trust between team members positively affects the productivity of OSS projects (Xu and Jones, 2010; Stewart and Gosain, 2006; Lane et al., 2004). The importance of trust was also validated in a recent study that showed that before accepting a code change, OSS project owners form an opinion about the trustworthiness of a code author based on information in the author’s profile (Marlow et al., 2013).

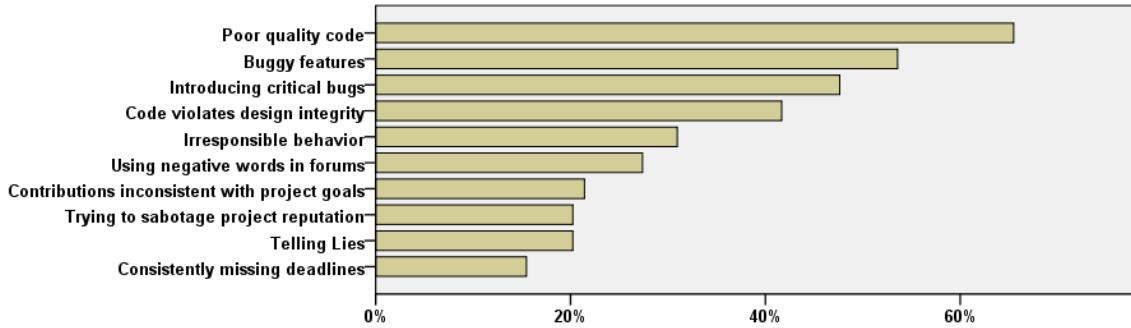


Figure 2.6: Factors that cause a peer to lose trust

2.5.4 RQ2: Losing trust in a peer

We asked the respondents to indicate which of the following 11 factors caused them to lose confidence in a peer (2.A- Q14). Post-hoc, during the analysis process, we grouped the 11 factors as follows:

1. **Coding-related factors:** Poor quality code, buggy features, introducing critical bugs, and codes violate design integrity;
2. **Professional factors:** Contributing code not consistent with project goals, trying to sabotage project reputation, contributing to a rival project, and consistently missing deadlines; and
3. **Interpersonal factors:** Telling lies, irresponsible behavior, and using negative words in forums.

As shown in Figure 2.6, the **Coding-related** factors were the top causes for losing trust in a peer, followed by most of the **Interpersonal** factors.

To further understand trust, we asked whether it was possible for a peer to regain trust once confidence had been lost and why or why not (2.A-Q15). Of the 80 respondents who answered this question, 68 (85%) believed it was possible to regain lost trust, 7 (8%) believed it was not possible to regain lost trust and the remainder had not experienced lost trust.

Of those who said it was possible to regain trust, 77% stated that regaining trust is only possible through work. In addition, 16% stated that effective communication is required to regain trust. A peer who has lost trust has to admit the fault, reverse the action that resulted in

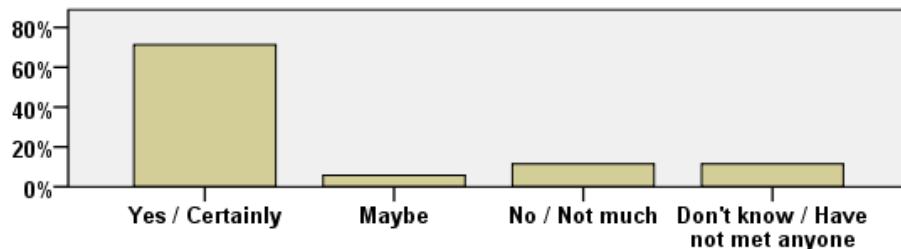


Figure 2.7: Participants' belief that meeting in person affects their impressions

lost trust (if possible), demonstrate commitment to the community, and produce quality work. Some respondents said that regaining trust is very difficult and takes a long time.

2.5.5 H3: Effect of Meeting in Person on Peer Impression

The qualitative analysis of the 87 responses to the question about the effects of meeting a peer in person (2.A-Q11) resulted in the four responses shown in Figure 2.7. The respondents overwhelmingly viewed meeting in person to have a positive impact on peer impression (71% vs. 11%). This result supports Hypothesis 3 and the findings from Crowston et al. (2007). The following quotes help describe some of the reasons for the positive responses. First, many respondents mentioned that meeting in person helps to humanize the peer. For example:

“Meeting a person in real life allows one to identify with them as an individual and develop a rapport (or not!). Until then they are merely a name, and one’s impression of them is solely based upon the words they have written in communications (mail, IRC) and in code (patches, commits). I have found that such meetings are generally beneficial.”

Second, many respondents mentioned that meeting in person improved communication because they could better understand the messages. As one respondent said,

“We have a meeting every year, and it can make a lot of difference to later be able to “hear” what their emails say as if they were speaking. For example, one person turned out to be a very funny person in real life, but did not always understand that their jokes do not go over well in written form. Having met them, it was easier to understand when they’re joking, and so get less upset when they say something supposedly funny.”

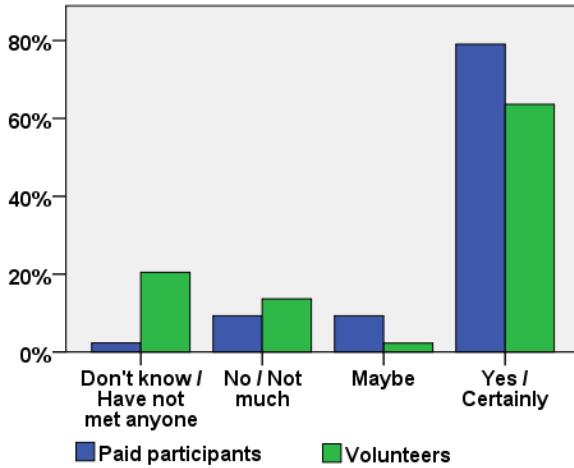


Figure 2.8: Opinions regarding meeting in person, paid participants vs. volunteers

Another respondent said,

"It makes further communication and interaction easier. I've dealt with more people that I haven't met and our interactions are perfectly fine, but the ones I've met in person are a bit more loose."

Third, some respondents mentioned that after meeting in person they felt a stronger sense of association and were able to assume a positive intent from email communications. As one respondent said,

"I've found that people (including myself) tend to do a better job of assuming positive intent from people that they've met in person."

Figure 2.8 illustrates that the effects of meeting a teammate in person were significantly different depending on the type of participant ($\chi^2_{3,87} = 9.18, p = .027$). Two key observations emerge from this data. First, approximately 20% of the volunteer participants answered "Don't know/ Have not met anyone," compared with only 2% of the paid participants. Second, 80% of the paid participants answered "Yes / Certainly," compared with only 60% of the volunteer participants.

To better understand this result, we hypothesized that the paid participants would be more likely to work in the same office as other project members (who were likely also paid by the same organization to participate in the project). This co-location should help facilitate in person

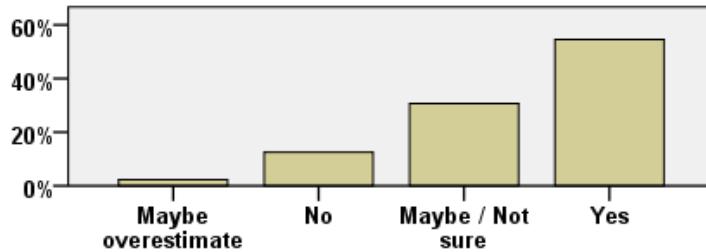


Figure 2.9: Participants’ belief that peers have accurate impressions

interactions. However, when we tested this hypothesis, we found no significant relationship between participant type and project distribution (Section 2.4.4).

Therefore, we hypothesize an alternate explanation for this result. Companies that pay employees to participate in OSS projects also pay for those participants to attend project meetings and conferences, providing more opportunities to meet their peers in person. Conversely, a volunteer participant will typically have to pay his or her own expenses to attend project meetings or conferences. Volunteers are likely able to attend fewer conferences and project meetings than paid participants. Moreover, small scale or unsuccessful OSS projects, whose members tend to be volunteers, might not be able to arrange a conference at all. For these reasons, paid participants likely have more opportunities than volunteers to meet their peers. The survey data supports this conclusion as it shows that it is more common for volunteer participants to never meet any peers in person. Approximately 20% of the volunteer participants said that they have never met anyone in person, compared with only 2% of the paid participants. Because the paid participants meet their peers in person more frequently, more of them have a basis upon which to state that meeting in person affected their impression of their peers. We plan to test this hypothesis more specifically in future surveys.

2.5.6 H4: Accuracy of Peer Impression

The qualitative analysis of the 88 responses to 2.A-Q10 resulted in four categories of answers. Figure 2.9 shows that 55% of the respondents believed their peers have an accurate impression of their abilities, 33% were unsure, but leaning positive, and only 13% believed that their peers do not have an accurate impression of their abilities. These results support Hypothesis 4.

Some specific quotes from the responses help to explain this result. Many respondents

mentioned that because in OSS communities everything is archived and open, participants who maintain regular collaboration with the community can easily get a sense of a participant's ability. For example, one respondent said,

*"Yes. A lot of the communication happens in the open *and* is archived in public places. The openness of discussions means not only that a lot of people can see what is going on, but also that everyone is on the same boat regarding what happened, what was discussed, what conclusions we reached, and what we are currently doing."*

Some respondents mentioned that the collaborative development process is helpful for building peer impressions. For example, one respondent said,

"Yes. Code reviews are a daily metric."

Conversely, some of the reasons given for inaccurate impressions include: not being able to spend enough time and not being able to use full capabilities. As one respondent said,

"Probably not. There has never been a need to use my full capability. I just contribute in areas that are productive, and leave areas that I could work in when there are other bodies who can do that work."

Surprisingly, whether a participant was paid or volunteer did not affect whether he/she thinks his/her peers have an accurate view of his/her abilities. We expected that more paid participants would think their peers (who may be co-located in the same company) have an accurate view of them than the volunteers (who are distributed and may never meet) would. We will investigate this result further in future studies.

2.5.7 H5: Effect of Project Contributions on Opinions about Productivity and Competency

We asked the respondents to rate the importance, on a 5-point scale, of the following 13 factors for judging whether a peer is productive or competent (2.A-Q8)

1. **Work style:** creativity, accuracy, response time and efficiency;
2. **Historical factors:** A peer's handle (i.e., screen name or loginId), background and online profile; and

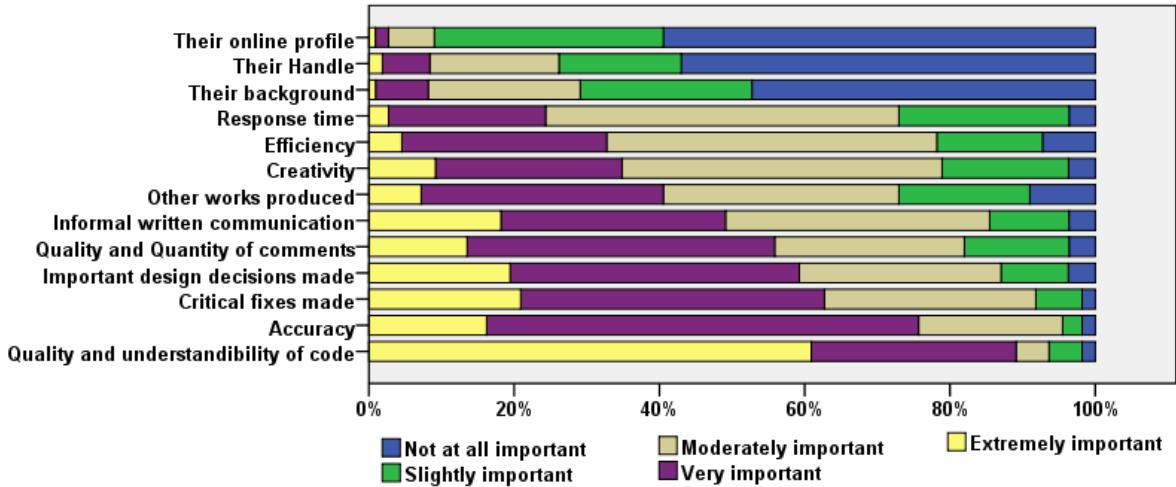


Figure 2.10: Importance ratings of the factors to decide a peer as productive or competent

3. Project Contribution factors: Quality and understandability of committed code, quality and quantity of comments, critical fixes made, important design decisions made, informal written communication and other work produced.

Figure 2.10 shows the 13 factors sorted in increasing order according to the proportion of the top two ratings (Extremely important and Very important). The median for each factor can be identified by observing which category the 50% line crosses through.

The **Historical** factors appear at the top of the chart, indicating that the respondents found these factors to have relatively low importance for judging a peer to be productive. Next, three of the four **Work Style** factors appear, indicating moderate importance of these factors. The fourth **Work Style** factor, *accuracy*, is the second most important factor overall. Finally, the **Project Contribution** factors appear at the bottom of the chart, indicating them to be the most important overall. Even though *accuracy* is a **Work Style** factor, it could also have been considered a **Project Contribution** factor because accuracy affects the quality of a participant's contribution. These results suggest that overall, OSS participants consider **Project Contribution** to be more important than **Work Style** or **Historical** factors for judging the productivity or competency of a peer. These results support Hypothesis 5.

This result supports the findings of Marlow et al. (2013) that general coding ability, project relevant skills, and interaction style are the most important factors for forming impressions about the competency of a peer. The participants mentioned the use of various cues to form im-

pressions, such as: amount of activity, frequency of commits, length of contribution, languages used, and discussion threads.

The importance of **Project Contribution** factors is also not surprising. OSS communities are often described as ‘meritocracies’ (Fielding, 1999; Raymond, 1998; Schmidt and Porter, 2001) (the form of governance where responsibilities are assigned to individuals solely based upon their merits). Because committed code includes the author names, the quality and understandability of each participant’s code can be easily judged by his or her peers. Understandable code also helps to facilitate collaboration. Unsurprisingly, this factor was chosen as the most important factor.

Furthermore, OSS participants typically discuss important design decisions and critical bugs in mailing lists and in IRC. Therefore, all developers who subscribe to the list or participate in the IRC are able to judge the intellect of those developers that provide the solutions. Because the success and reputation of an OSS project are critically dependent upon developers making good design decisions and fixing critical bugs, those factors are very important for judging whether a peer is productive.

Figure 2.2 shows that approximately 19% of the respondents thought their peers were “competent or dedicated” (RQ1 - Section 2.5.1). Figure 2.11 shows that respondents who found their peers “competent or dedicated” gave significantly higher importance to the two most important factors in Figure 2.10, “Quality and understandability of code” (Mann-Whitney $U=617.0$, $Z=-2.53$, $p=.011$) and “Accuracy” (Mann-Whitney $U=612.0$, $Z=-2.593$, $p=.01$). Relative to the peer impression formation process, this result suggests that when a participant performs a significant task (e.g., developing a critical module) with quality, understandability and accuracy, his/her peers form a better impression.

2.5.8 RQ3: Factors affecting Ease or Difficulty of Working Together

The responses to Question 12 (Appendix A), shown in Figure 2.12, illustrate that the respondents viewed five of the six factors as very influential in judging how easy it is to work with a peer. Conversely, the responses to Question 13 (Appendix A), shown in Figure 2.13, illustrate how the respondents thought 10 of the 11 factors from RQ2 affected their judgement of how difficult it was to work with a peer. In the same way as for RQ2, post-hoc we grouped

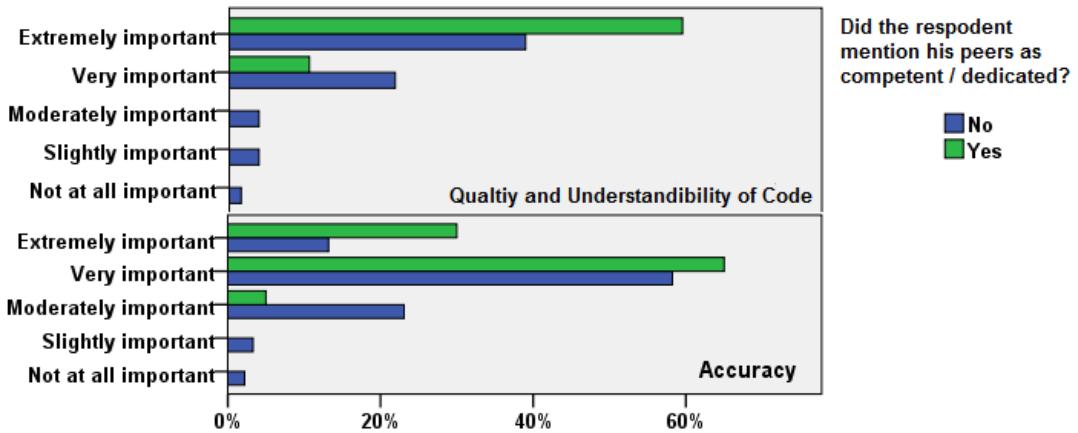


Figure 2.11: Differences in distribution based on respondents' impressions about peers

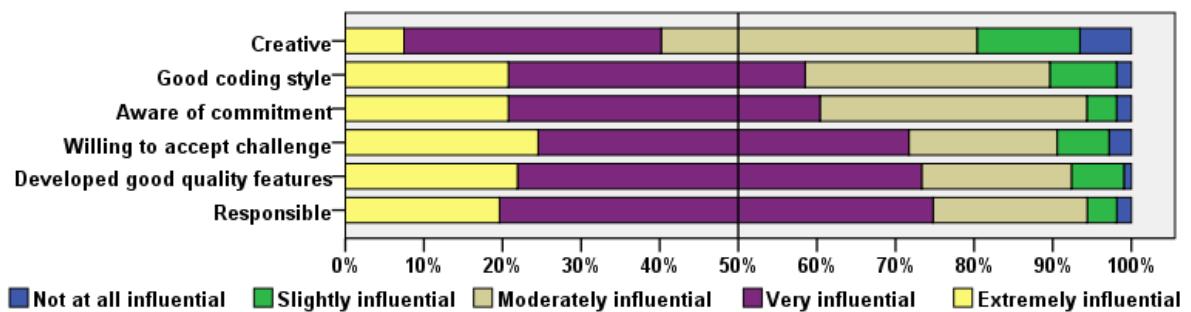


Figure 2.12: Factors influencing how easy it is to work with a peer

these factors into three categories. Not surprisingly, the results are similar to those for RQ2. The **Coding-Related** factors were the top reasons for deciding it was difficult to work with a peer, followed by most of the **Interpersonal Factors**. As a group, the **Professional Factors** were the least influential. The similarity of the results for RQ2 and RQ3 suggests that OSS participants may start to lose confidence in a peer when they find it difficult to work with that peer. This hypothesis will be tested in future studies.

The fact that **Coding-Related** factors were the most important is not surprising. Because OSS participants generally collaborate at the code level, and often cannot have FTF meetings, understandable code is of utmost importance. Moreover, due to the lack of formal requirements and documentation in most OSS communities, the quality and understandability of source code is very important for effective collaboration and future maintenance. Therefore, OSS participants find it difficult to work with team members who write poor quality code that has bugs or violates design integrity.

One interesting observation from Figure 2.13 is that most of the respondents (70%) indi-

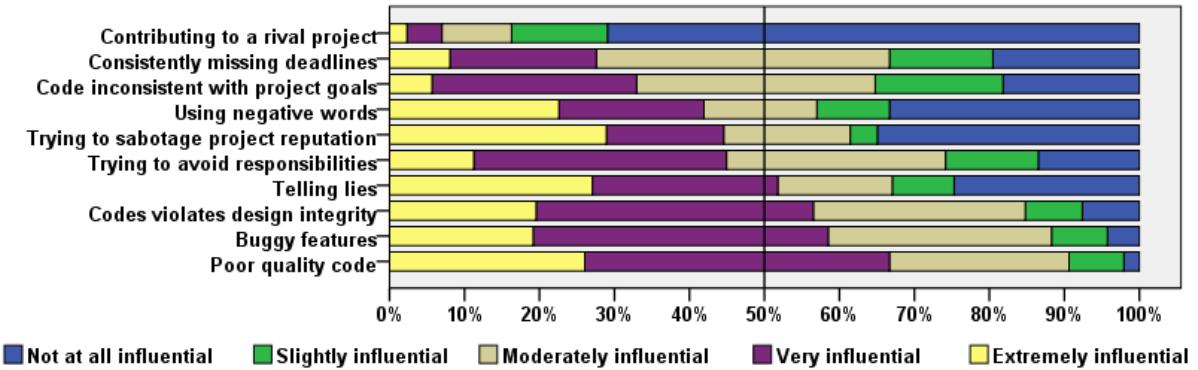


Figure 2.13: Factors influencing the decision to judge a peer difficult to work with

cated that contributing to a rival project did not have a negative influence. It is possible that OSS participants consider working on a rival project to be advantageous by helping them become aware of the strong and weak features of the project. The experiences from rival projects may also help guide development directions.

2.6 Threats to Validity

As with any empirical study there are some threats to validity that need to be discussed. This section is organized around three common types of validity threats.

2.6.1 Internal validity

A threat to internal validity for our study is related to the selection of study participants. We sent the survey requests only to successful OSS project communities. Those communities might not be a good representation of the overall OSS world. Although we requested only active OSS participants to respond, we could not ensure this fact. However, we believe there was no motivation for inactive participants to respond to the survey because we did not offer any financial incentive. While analyzing the qualitative data, we suspected that four respondents may have answered the questions carelessly (i.e., they provided random texts in the open-ended responses). We excluded these responses from the analysis.

2.6.2 Construct validity

First, we selected the factors to include in various questions based on a reading of the literature and our own personal experiences. The lists of factors may not be complete. To combat this

threat, we provided respondents with an “Others (Please specify)” option. However, very few respondents selected the “Other” option. For those that did, no particular factor was mentioned frequently. Therefore, we believe this threat is not significant.

Second, we did not perform a formal validity analysis of the survey. Rather, we had the survey questions reviewed by survey and domain experts to ensure their understandability and to remove any potential bias. Because the results of the survey were largely consistent with findings from earlier studies, we do not believe this threat is serious.

2.6.3 External validity

We are not able to definitively establish that our sample is representative of the entire OSS population. While OSS communities vary a lot based on product, participant type, community structure, and governance, our sample represents a large number of OSS projects. Even though we did not take into account most of those factors, we do have confidence that our sample is representative. Of course, the results may not be applicable to all OSS communities.

2.7 Conclusion

This chapter presents the results of a survey that explored the peer impression formation process in OSS communities. The results showed that the majority of survey respondents had positive experiences when collaborating with their peers. The respondents consider the perceived expertise and trustworthiness of their peers to be very important factors for interaction and impression formation. Although meeting a peer FTF is infrequent, those meetings can positively affect peer impression. Factors related to project contributions are the most important when forming an impression of whether a peer is productive and how easy or difficult it is to work with a peer.

The Psychology literature suggests that impression formation depends on the familiarity with personal qualities. In OSS projects, participants are not aware of most of the personality traits of their peers. Therefore, OSS participants become very task-focused when they form impressions about a peer. The survey results indicated that the respondents placed the lowest emphasis on interpersonal factors and the highest emphasis on project contributions when forming impressions of their peers.

The introduction mentioned the four domains of CMC interaction (i.e., relative anonymity, reduced importance of physical appearance, attenuation of physical distance, and greater control over the time and pace of interactions) proposed by McKenna and Bargh (2000). The current survey results address two of those domains. We found that collaboration between OSS participants is relatively anonymous and they give the least importance to the background and profile of a peer. However, the results do not support reduced importance of physical appearance. Although OSS participants rarely meet their peers in person, this does not imply that they do not want to meet in person or that meeting in person does not benefit them. In fact, the results show that meeting in person improves communication by helping participants better understand the tone of a peer's messages and helps to humanize their peers.

In this chapter, we studied components of impression formation between OSS participants and found some evidences regarding how those components form and evolve. The results provide some insight into the impression formation process between OSS peers. Impression formation is different and complicated in virtual organizations like an OSS community compared with traditional organizations. Yet, OSS participants consider those impressions very important for communication and collaboration. Therefore, we believe peer impressions will also affect the outcomes of OSS projects. In the future, we plan to study how project outcomes are affected by peer impressions and how to improve the impression formation process in OSS organizations.

2.8 References

- Cialdini, R. B. (2009). *Influence: Science and Practice*. Pearson Education Canada, 5th ed. edition.
- Crowston, K., Howison, J., Masango, C., and Eseryel, U. Y. (2007). The role of face-to-face meetings in technology-supported self-organizing distributed teams. *IEEE Transactions on Professional Communication*, 50(3):185–203.
- Evans, P. and Wolf, B. (2005). Collaboration rules. *Harvard Business Review*, 83(7):96–104.

- Fielding, R. T. (1999). Shared leadership in the apache project. *Communications of the ACM*, 42(4):42–43.
- Gacek, C. and Arief, B. (2004). The many meanings of open source. *IEEE Software*, 21(1):34–40.
- Gallivan, M. J. (2001). Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. *Information Systems Journal*, 11(4):277–304.
- Ghosh, R. A., Glott, R., Krieger, B., and Robles, G. (2002). Free/libre and open source software: Survey and study. Technical report, International Institute of Infonomics.
- Green, M. C. (2007). *Trust and social interaction on the Internet*. Oxford University Press.
- Guadagno, R. and Cialdini, R. (2002). Online persuasion: An examination of gender differences in computer-mediated interpersonal influence. *Group Dynamics: Theory, Research, and Practice*, 6(1):38–51.
- Guadagno, R. and Cialdini, R. (2005). *Online persuasion and compliance: social influence on the Internet and beyond*, pages 91–113. Oxford University Press.
- Gutwin, C., Penner, R., and Schneider, K. (2004). Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, CSCW ’04, pages 72–81.
- Hars, A. and Ou, S. (2002). Working for free? motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6(3):25–39.
- Hertel, G., Niedner, S., and Herrmann, S. (2003). Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32(7):1159–1177.
- Holton, J. A. (2001). Building trust and collaboration in a virtual team. *Team Performance Management*, 7(3/4):36–47.

- Jarvenpaa, S. L. and Leidner, D. E. (1998). Communication and trust in global virtual teams. *Journal of Computer-Mediated Communication*, 3(4):791–815.
- Jensen, C. and Scacchi, W. (2005). Collaboration, leadership, control, and conflict negotiation and the netbeans.org open source software development community. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS '05, page 196b.
- Kayworth, T. and Leidner, D. (2000). The global virtual manager: a prescription for success. *European Management Journal*, 18(2):183–194.
- Kenny, D. A. (1994). *Interpersonal perception: A social relations analysis*. Guilford Press.
- Lakhani, K. R. and Wolf, R. G. (2005). *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. MIT Press.
- Lane, M. S., Glen, V., and Basnet, P. (2004). Trust in virtual communities involved in free/open source projects: an empirical study. In *Proceedings of the 15th Australasian Conference on Information Systems*, ACIS'04.
- Markus, M. L., Manville, B., and Agres, E. (2000). What makes a virtual organization work? *Sloan Management Review*, 42(1):13–26.
- Marlow, J., Dabbish, L., and Herbsleb, J. (2013). Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer Supported Cooperative Work*, CSCW '13, pages 117–128.
- Maznevski, M. L. and Chudoba, K. M. (2000). Bridging space over time: Global virtual team dynamics and effectiveness. *Organization Science*, 11(5):473–492.
- McKenna, K. Y. and Bargh, J. A. (2000). Plan 9 from cyberspace: The implications of the internet for personality and social psychology. *Personality and Social Psychology Review*, 4(1):57–75.
- Moore, C. D. (2007). *Impression Formation*, volume 2. Blackwell Publishing.

- Moorman, C., Deshpande, R., and Zaltman, G. (1993). Factors affecting trust in market research relationships. *The Journal of Marketing*, 57(1):81–101.
- Peters, L. M. and Manz, C. C. (2007). Identifying antecedents of virtual team collaboration. *Team Performance Management*, 13(3/4):117–129.
- Potter, R. E., Cooke, R. A., and Balthazard, P. A. (2000). Virtual team interaction: assessment, consequences, and management. *Team Performance Management*, 6(7/8):131–137.
- Raymond, E. (1998). Homesteading the noosphere. *First Monday*, 3(10-5).
- Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49.
- Roberts, J. A., Hann, I.-H., and Slaughter, S. A. (2006). Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Management Science*, 52(7):984–999.
- Schmidt, D. C. and Porter, A. (2001). Leveraging open-source communities to improve the quality performance of open-source software. In *Proceedings of the First Workshop on Open-Source Software Engineering*.
- Stewart, K. J. and Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly*, 30(2):291–314.
- Tanis, M. and Postmes, T. (2003). Social cues and impression formation in cmc. *Journal of Communication*, 53(4):676–693.
- Treude, C. and Storey, M. (2010). Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering*, ICSE ’10, pages 365–374.
- Von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241.

Xu, B. and Jones, D. R. (2010). Volunteers' participation in open source software development: a study from the social-relational perspective. *SIGMIS Database*, 41(3):69–84.

Zand, D. E. (1972). Trust and managerial problem solving. *Administrative Science Quarterly*, 17(2):229–239.

APPENDIX

2.A Survey questions

Q1. What open source project are you most actively involved in?

Q2. What other open source project(s) have you actively worked on?

Q3. Please name or describe your roles on the open source project? (e.g., add new code, report bugs, fix bugs, maintain project infrastructure, make strategic decisions about direction of projects, etc...)

Q4. Please briefly provide your thoughts on the quality of the software.

Q5. Please briefly provide your thoughts about working with other developers on the project.

Q6. Do you contribute to the open source project during your regular job, or only in your spare time?

- I contribute during my regular job (getting paid)
- I only contribute in my spare time (volunteer)

Q7. How are the project contributors distributed?

- Contributors are roughly evenly distributed across many ($>= 5$) different organizations
- Most contributors come from a small number (< 5) of organizations
- Most contributors are members of one organization

Q8. When deciding if a teammate is productive or competent, how important are each of the following factors? Please rate each factor on a 1-5 scale. (1= Not at all important, 2=Slightly important, 3=Moderately important, 4=Very important, 5=Extremely important)

- Creativity
- Accuracy
- Response time
- Efficiency
- Their handle
- Their background
- Their online profile
- Quality and understandability of the code committed to repository
- Quality and quantity of comments in their code
- Other work products produced (i.e. design documents, test cases)
- Informal written communication (i.e. post to mailing lists, IRC chats)
- Critical fixes
- Important design decisions made
- Others(Please specify) -----

Q9. Thinking about your interactions with your teammates, on a scale of 1-5 how important is each of the following factors? (1= Not at all important, 2=Slightly important, 3=Moderately important, 4=Very important, 5=Extremely important)

- Your impression of their level of expertise
- How much you trust them

Q10. Do you think your teammates have an accurate view of your abilities? Why or why not?

Q11. Does meeting a teammate in person affect your impression of them? Explain.

Q12. Think of a time when you found a teammate easy to work with. How important were the following factors in deciding that teammate was easy to work with? (1= Not at all important, 2=Slightly important, 3=Moderately important, 4=Very important, 5=Extremely important)

- Creative
- Responsible
- Good coding style
- Willing to accept challenges
- Developed good quality features
- Others(Please specify)-----

Q13. Think of a time when you found a teammate difficult to work with? Which factors influenced your decision? (1= Not at all influential, 2=Slightly influential, 3=Moderately influential, 4=Very influential, 5=Extremely influential)

- Telling lies
- Trying to avoid responsibilities
- Consistently missing deadlines
- Poor quality code
- Buggy features
- Codes violating design integrity
- Trying to sabotage project reputation
- Contributing to a rival project
- Using negative words in forums (e.g., swear words, insults, racial remarks etc.)
- Contributing code that is not consistent with project goals
- Others(Please specify)-----

Q14. Have you ever lost confidence in a teammate to the point where you would not trust their opinions (e.g., on design decisions), or the quality of their work products (e.g., code produced)?

What factors that caused you to lose this confidence? (Please check all applicable)

- Telling lies
- Consistently missing deadlines
- Poor quality code
- Introducing critical bug
- Buggy features
- Code violates design integrity
- Trying to sabotage project reputation
- Contributing to a rival project
- Irresponsible behavior
- Using negative words in forums
- Contributing code that is not consistent with project goals

Q15. Is it possible for a teammate to regain trust once they have lost your confidence? If so, how?

CHAPTER 3

INSIGHTS INTO THE PROCESS ASPECTS AND SOCIAL DYNAMICS OF CONTEMPORARY CODE REVIEW

In recent years, many open source software (OSS) and commercial projects have adopted peer code review (Bacchelli and Bird, 2013), a practice, where developers subject their code to scrutiny by their peers. While the underlying concepts of *contemporary code review* (Rigby and Bird, 2013) are similar to the traditional Fagan inspection (Fagan, 1976), there are also marked differences. A Fagan inspection is a heavyweight process requiring synchronous meeting between the participants in multiple phases. Conversely, contemporary code review is lightweight, more informal, asynchronous, and supported by specialized tools. Despite studies that indicate Fagan inspections are effective for software quality improvement (Fagan, 2002), their typically high cost and formality have prevented widespread adoption (Johnson, 1998; Votta Jr, 1993). Conversely, contemporary code review has addressed many shortcomings of Fagan inspection and has shown increasing adoption in industry and OSS contexts (Balachandran, 2013; McIntosh et al., 2014; Rigby and Bird, 2013).

Because many OSS projects, e.g. Apache(Rigby et al., 2008), Chromium¹, Mozilla², Qt³, and Android⁴, now require peer-review prior to merging new code into the main project codebase, there are a large number of developers regularly participating in code reviews. From the industrial perspective, Google⁵ and Facebook⁶have adopted mandatory code reviews and approximately 50,000 Microsoft developers actively practice code reviews (Bosu et al., 2015). Our recent survey found that developers spend approximately six hours per week in code review (Bosu and Carver, 2013). Given the large number of developers who practice code review,

¹ <https://www.chromium.org/developers/contributing-code>

² <https://www.mozilla.org/hacking/reviewers.html>

³ https://wiki.qt.io/Qt_Contribution_Guidelines

⁴ <https://source.android.com/source/life-of-a-patch.html>

⁵ <http://matt-welsh.blogspot.com/2012/02/my-love-affair-with-code-reviews.html>

⁶ <https://framethink.wordpress.com/2011/01/17/how-facebook-ships-code/>

the total time devoted to code review is quite significant. Therefore, increasing the effectiveness of contemporary code review can greatly improve software development productivity.

To improve a process or practice, empirical researchers suggest three-step approach: first, understanding the current process to identify improvement opportunities; second, evaluating current process and new ideas; and finally, improving the process by incorporating suggestions (Wohlin et al., 2003). Based on this three-step framework, the high level objective of this study is to *better understand the contemporary code review process and its benefits*.

The specific goal of this study is to gather information about 1) the code review process, 2) developers' expectations from code review, and 3) how code review impacts developers' impressions of their peers. On this goals, we rigorously designed and validated a survey instrument. We sent the survey to code review participants from 36 popular OSS projects and from Microsoft, receiving 287 and 416 responses, respectively.

The primary contributions of this study are:

- Better understanding of developers' perception about contemporary code review;
- Better understanding of why and how developers collaborate during code reviews;
- Empirical evidence regarding non-technical benefits of code reviews;
- A comparison of code review practices between OSS and Microsoft projects; and
- Illustration of systematically designing and analyzing a software engineering (SE) survey.

The remainder of the paper is organized as follows. Section 3.1 provides a brief description of contemporary code review process and prior literature on code reviews. Section 3.2 defines the research questions. Section 3.3 describes the research method. Section 3.4 characterizes the study participants. Section 3.5 provides the results. Section 3.6 discusses the implications of the results. Section 3.7 describes the threats to validity. Finally, Section 3.8 provides directions for future work and concludes the paper.

3.1 Background

This section provides a brief background and prior research on contemporary code review.

3.1.1 Contemporary Code Review Process

One key aspect of contemporary code review is that it is tool-based. Some popular code review tools include: Gerrit⁷, Phabricator⁸, and ReviewBoard⁹. Figure 3.1 provides a simplified overview of the contemporary code review process. First, the author creates a *patch-set* (i.e. all files added or modified in a single revision) along with a description of the changes, and submits that information to the code review tool. Then the author (or someone else) selects the reviewer(s) for the patch-set. The code review tool then notifies the reviewer(s) about the incoming review. During the review process, the tools highlight the changes between revisions in a side-by-side display. The reviewers and the author can insert comments into the code. After the review, the author can address the comments and upload a new patch-set to initiate a new review iteration. This review cycle repeats until either the reviewers approve the changes or the author abandons the change. If the reviewers approve the code, then the author can commit it to the project repository.

3.1.2 Overview of Contemporary Code Review Research

In recent years, there have been several studies on code review practices. Most of those studies focused on understanding contemporary code review processes in different projects. Rigby has published a series of studies examining informal peer code review practices in OSS projects (Rigby and German, 2006; Rigby et al., 2008; Rigby and Storey, 2011), and comparing the review process between commercial and open source projects (Rigby and Bird, 2013). To characterize the code review practices, (Rigby and German, 2006) proposed a set of code-review metrics (i.e. acceptance rate, reviewer characteristics, top reviewer vs. top committer, review frequency, number of reviewers per patch, and patch size). Other researchers calculated similar metrics for five OSS projects and concluded that code review practices vary across different OSS projects based on age and culture of the projects (Asundi and Jayant, 2007).

⁷ <https://code.google.com/p/gerrit/>

⁸ <http://phabricator.org/>

⁹ <https://www.reviewboard.org/>

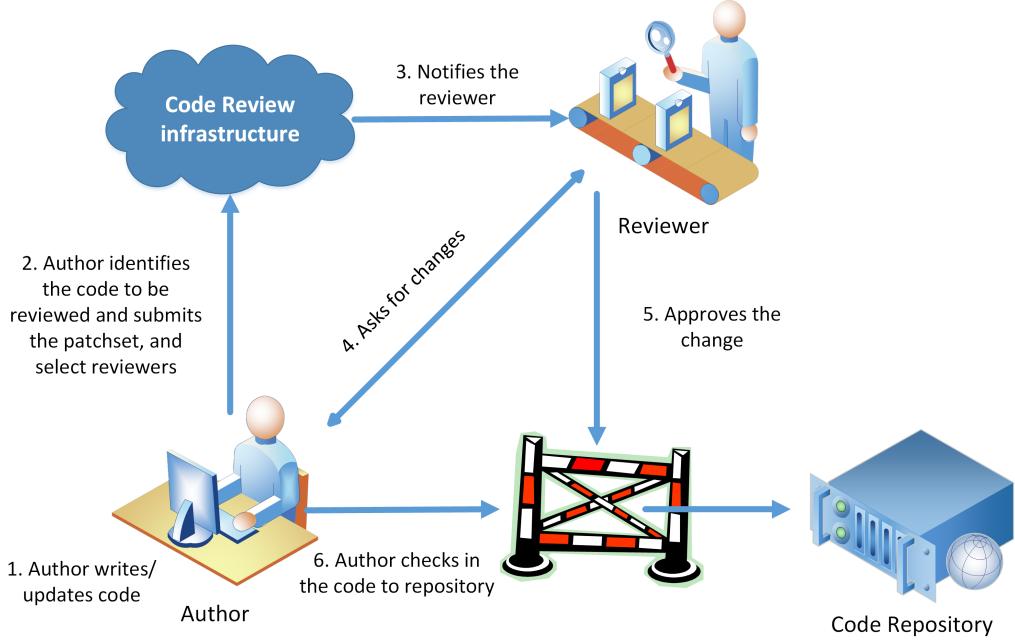


Figure 3.1: Simplified code review workflow

However, these findings were contrasted in a later study, which found that despite large differences among five OSS projects and several commercial projects, their code review metrics were largely similar (Rigby and Bird, 2013). While those studies provide several key insights to better understand of contemporary code review, no study has explored several key areas (e.g., developers perception of the importance of code reviews, social interaction during code reviews, and best strategies to assist poor quality code). A better understanding of these unexplored areas can help project managers decide upon the usefulness of code reviews and help developers improve their review process.

After seeing the successful adoption of code review practices by OSS projects, many commercial organizations have recently adopted peer code review practices (Sutherland and Venolia, 2009; Rigby et al., 2012; Bacchelli and Bird, 2013; Balachandran, 2013). Contrary to OSS projects, code review participants at Microsoft use both synchronous and asynchronous communication mediums for code reviews and consider communications during code reviews essential to understand code changes and design rationale later. Microsoft developers expressed a need to retain code review communications for later information needs (Sutherland and Venolia, 2009). Another study at Microsoft found that although finding defects is a primary motivation for code reviews, other benefits (e.g. knowledge dissemination, team awareness, and

identifying better solutions) may be more important. The major challenge is understanding the code changes (Bacchelli and Bird, 2013). While these studies characterized the code reviews in commercial projects, only one study (Rigby and Bird, 2013), which focused on quantitative aspects of code reviews, has compared and contrasted between the code review practices of OSS and commercial projects. Since developers' motivations and project governance differ between OSS and commercial organizations, code review collaborations may also differ between OSS and commercial projects.

While most of the earlier exploratory studies focused on understanding the code review practices, a few of the recent studies have focused on understanding the impact of different factors on code reviews. Code review characteristics, such as review size, component, priority, organization, reviewer characteristics, and author experience significantly influence on both review completion time and outcome (Baysal et al., 2013). Moreover, a reviewer's prior experience in changing or reviewing the artifact under review and project experience increases the likelihood of providing useful feedback (Bosu et al., 2015). While these studies focused on technical human factors and characteristics of the code changes, no studies have focused on the non-technical human factors (i.e., author's reputation, and relationship between an author and a reviewer). Because code review facilitates direct collaboration between people, a better understanding of the impacts of various human factors is crucial to improve the code review process.

A few recent studies have investigated various technical benefits of code reviews. Although the primary goal of code reviews is defect detection, because three-fourths of the review comments are related to maintainability issues code review may be more beneficial for projects which require highly maintainable code (Beller et al., 2014). Code reviews have significant impact on software quality. A recent study found that both low code review coverage (i.e., the proportion of changes that have been reviewed) and low review participation (i.e., the number of reviewers) often increase the likelihood of post-release defects (McIntosh et al., 2014). While these studies focused on the technical benefits of code review, only one study (Bacchelli and Bird, 2013) has explored the non-technical benefits of code reviews. The evidence about the non-technical benefits (i.e., impressions formation, knowledge sharing, and mentoring) has

been mostly anecdotal. Empirical evidence regarding various benefits of code reviews can encourage project managers to adopt code reviews for their projects.

3.2 Research Questions

The goal of this study is to better understand the contemporary code review process and its benefits. This study explores six research questions. The remainder of this section defines each question.

3.2.1 Importance of Code Review

Code reviews require significant effort and delay merging of code to the main branch by 1-2 days (Rigby and Bird, 2013). However, recent studies indicate that only one-fourth of code review comments relate to functional defects (Beller et al., 2014; Czerwonka et al., 2015), which raises questions whether developers perceive the effort spent in code review as beneficial. To better understand how developers view the importance of code review, the first research question is:

RQ1: *Why do developers consider code reviews important (or not important) for their projects?*

3.2.2 Code Review Process

Because projects often mandate the use of code review, developers spend a significant amount of time performing code reviews. To quantify this effort, the second research question is:

RQ2: *How many hours, on average, developers spend in code reviews?*

In our prior study of code review-based social networks in OSS projects, we observed the presence of sub-communities and a higher volume of interactions between some developer pairs (Bosu and Carver, 2014a). Subsequently, we found that experienced developers received more timely feedback on review requests than newcomers (Bosu and Carver, 2014b). These results suggest that a history of interactions may influence a reviewer's acceptance and prioritization of particular reviews. The next research question investigates this phenomenon.

RQ3: *How do developers decide to accept or reject an incoming code review request?*

Reviewers may use different criteria to determine whether a code change is of high quality. For example, reviewers may have different opinions on the effects of coding style on quality (Bosu et al., 2015). The next research questions seeks to better understand these factors:

RQ4: *Which code characteristics are indicative of low quality code?*

The goal of code review is not only to identify issues in a code change but also to help the author resolve those issues. Experienced reviewers can mentor code authors regarding coding techniques, project design, or API usage. The next question seeks to better understand this process:

RQ5: *How do reviewers help low quality code improve to the level required for inclusion in the project?*

3.2.3 Impact of Code Review on Peer Impressions

The intense interactions during the code review process allow participants the opportunity to gain a unique insight into the abilities of their peers. For example, if a reviewer repeatedly finds the contributions of a particular code author to be of high quality, the reviewer may consider that author to be highly competent or intelligent. As a result, code review collaborations may help the participants form accurate perceptions of each other. Moreover, a reviewer may be more likely to trust project-related decisions made by an author known to be competent. Because some of the primary benefits of contemporary code review are non-technical, i.e. beyond defect detection (Bacchelli and Bird, 2013), it is important to understand what those non-technical benefits may be. To help identify these benefits, we present three research questions. The first question, on the positive side:

RQ6: *How does the use of a high-quality or an outstanding problem-solving approach affect the reviewers' perception of the code author?*

Conversely, low-quality code may result in negative impressions. Therefore, the next question is:

RQ7: *How does a poorly written code affect the reviewers' perception of the code author?*

As addressed in previous research questions, code reviews could have either positive or negative impacts on the impressions that teammates have of each other. To help judge the value of code reviews, it is important to understand whether the overall effect of is positive or negative.

RQ8: *What is the effect of code review on peer impression?*

3.3 Research Method

We conducted two surveys to answer the research questions defined in Section 3.2. The first survey targeted OSS developers. We have published partial results of the first survey (Bosu and Carver, 2013). The second survey targeted developers from Microsoft. The remainder of this section describes the survey design process, participant selection criteria, pilot tests, data collection, and data analysis methods.

3.3.1 Survey Design

Because our goal was to measure peer impression constructs (i.e., RQ8), we followed well-regarded social and behavioral research methods to build scales (Fink, 2003; DeVellis, 2011). In this approach, rather than directly asking the participants about each of the constructs of interest, the researchers define a number of scale items that focus on different aspects of the same underlying construct. Then, during analysis, the researchers are able to gain a more complete understanding of the construct based on the diverse set of scale items.

To address the survey goals, we identified four key constructs. For each construct, the respondents indicated their perception of a series of statements (scale items). We drew these statements from well-established scales in psychology, information science, or organizational behavior. To ensure they were complete for software engineering, we added a few additional statements. The four constructs along with the sources for the statements are:

1. trustworthiness (Johnson-George and Swap, 1982; Rempel et al., 1985; Rotter, 1967; McAllister, 1995),
2. reliability (Rempel et al., 1985; Johnson-George and Swap, 1982; Robinson et al., 1991),
3. perception of expertise (Robinson et al., 1991), and

4. friendship (Bukowski et al., 1994; Robinson et al., 1991).

Table 3.5 in Appendix 3.9 lists statements for each construct. For each statement, the respondents used a 7-point scale to indicate whether it better described a *code review partner*¹⁰ or a *non-code review partner*¹¹. We defined the scale as follows: 1 = *describes a code-review partner and NOT a non-code review partner*, 4 = *describes both equally* and 7 = *describes a non-code-review partner and NOT a code-review partner*. To avoid any bias, the survey tool presented the statements in a random order without the name of the corresponding construct.

The survey also contained four multiple choice questions, 14 open-ended questions, and rating-scale question to address the research questions and gather demographics. Table 3.A in Appendix 3.9 lists the those additional questions (renumbered for the sake of simplicity). From now on, we will just refer the questions by question number.

3.3.2 Participant Selection

Developers must have participated in a sufficient number of code reviews before they would be able to accurately understand the code review process and the non-technical benefits of code review. In addition, for the effects of code review to impact impression formation, a developer must have participated in a number of these reviews. To ensure valid results, we only wanted to include developers with sufficient code review experience. To identify this set of developers for Survey 1, we developed a Java application to populate a local MySQL database with various type of information from the code review repositories of 34 OSS projects that used a code review tool described in Section 3.1.1. From that information, we identified developers who had participated in at least 30 code review requests (either as the author or the reviewer). Out of the 12,470 code review participants in the database, 2,207 matched this criteria. Similarly, for Survey 2, we queried Microsoft’s CodeFlow analytics platform (Bird et al., 2015) to select 2,000 developers who had participated in at least 30 code reviews. From these 2,000, we specifically recruited developers from two types of projects, 1) projects in which most developers are colocated, 2) projects in which most developers are distributed.

¹⁰ a person who reviews your code or whose code you review on a regular basis

¹¹ a person who has been a peer for some time, but you have rarely reviewed their code

3.3.3 Pilot Tests

To ensure the comprehensibility and validity of the scale items (statements) with respect to the constructs, we conducted five pilot tests. First, researchers from Psychology, Computer Science, and Management Information Systems reviewed the questions. This review led to several changes, including: (1) increasing the rating scales from 5- to 7-point, (2) rewording some questions to remove bias, and (3) adding questions for a broader perspective of the code review process.

Second, software engineering graduate students piloted the survey to identify any difficulties in understanding the questions and to estimate the time required to complete the survey.

Third, we sent the survey to 20 OSS code review participants from two projects in our database. The completion rate of this version of the survey was low. To address this problem, we rephrased some questions, reformatted the behavioral scale questions (Table 3.5) so they would appear less daunting, and reordered some questions.

Fourth, we sent the improved survey to 24 OSS code review participants from another project in our database. We received enough responses to analyze the internal consistencies of the four peer impression constructs. This analysis indicated that *reliability* scale had questionable internal consistency. Therefore, we added two questions to that construct.

Finally, we sent the survey to 117 OSS code review participants from 11 other projects. The completion rate was near 20% and the internal consistency of each construct was sufficient (Cronbach's $\alpha > 0.7^{12}$). Therefore, we deemed the survey ready for broad distribution. In three pilot studies with OSS participants, we asked for feedback and suggestions to improve the survey. We incorporated these suggestions into the final survey.

3.3.4 Data Collection

For Survey 1, in February 2013, we sent a survey invitation to each of the 2,046¹³ active code review participants in our database. Of those, 231 emails were undeliverable, leaving 1,815 valid invitations. Two weeks later we sent a reminder email. Approximately two weeks

¹² Chronbach's α is widely used to calculate the internal consistency of multiple-item measurements (e.g. the behavioral scale constructs). The results of this test are interpreted as follows: $\alpha \geq .9 \rightarrow excellent$, $\alpha \geq .8 \rightarrow good$, $\alpha \geq .7 \rightarrow acceptable$, and $\alpha < .7 \rightarrow questionable$.

¹³ 2,207 total less the 161 used in the pilots

Table 3.1: Coefficient Alpha (Cronbach's α) of the scales

Construct	OSS		Microsoft	
	α	Interpretation	α	Interpretation
Trust	.756	Acceptable	.798	Acceptable
Perception of Expertise	.811	Good	.839	Good
Reliability	.810	Good	.736	Acceptable
Friendship	.700	Acceptable	.758	Acceptable

later we closed the survey after the daily response rate decreased to almost zero. We received 287 responses (response rate of ~16%). For Survey 2, in September 2013, we sent a survey invitation to 2,000 Microsoft developers. After one week, we closed the survey. We received 416 responses (response rate of ~21%).

3.3.5 Data Processing and Analysis

The following subsections describe the data processing and analysis steps for the behavioral scale questions and the five open-ended questions.

3.3.5.1 Behavioral Scale Questions

We analyzed three forms of validity for the survey. First, we used expert researchers from Psychology, Management, and Computer Science to review the survey question for face validity (Litwin, 1995). Second, we carefully chose appropriate items from prior validated scales to ensure content validity (Litwin, 1995). Finally, we performed a principal components analysis with VARIMAX rotation to measure the construct validity (Litwin, 1995) of the scale items. Table 3.1 reports the α coefficients measures for the scales and their interpretations based on the two surveys.

This approach ensured high reliability and validity of the behavioral scales used in this study. In a prior article, we have detailed the reliability and validity measures of the survey instruments (Bosu and Carver, 2013).

3.3.5.2 Open-ended Questions

For the open-ended questions, we followed a systematic qualitative data analysis process. First, two analysts (Research Experience for Undergraduates (REU) students) extracted the general theme from each response to the OSS survey. Next, the first two authors, worked with those analysts to develop an agreed-upon coding scheme for each question. Using these coding

Table 3.2: Respondents' primary projects

Open Source Projects		Microsoft Projects	
Qt Project (36)	OpenStack (32)	Bing (54)	Windows (48)
CyanogenMod (28)	TYPO3 (20)	Office (43)	Azure (24)
Android (19)	MediaWiki (17)	Visual Studio (22)	XBox (13)
oVirt (17)	Linux kernel (16)	Ad Center (11)	Exchange (10)
Chromium OS (14)	Eclipse (9)	Dynamic AX (12)	Windows Phone (10)
Gromacs (9)	ITK (9)	IE (6)	SQL Server (6)
LibreOffice (8)	OpenAFS (6)	Dynamic CRM (4)	Sharepoint (4)
Scilab (5)	VTK (5)	Skype (3)	Halo (3)
Gerrit (4)	AOKP (2)	TFS (3)	HPC (3)
Couchbase (2)	Debian (2)	Autopilot (3)	Lync (3)
Python (2)	Others (22)	Kinect (2)	Others (129)

schemes, the two analysts independently coded the responses to one or more groups. After coding, they examined their results to identify any discrepancies. They discussed and resolved those discrepancies. For the Microsoft survey, the last two authors used the same process to analyze the qualitative data. For the five open-ended questions in the two surveys, we coded a total of 2626 responses.

3.4 Demographics

To provide proper context for the results, this section describes the demographics of the projects represented by the respondents and of the respondents themselves.

3.4.1 Projects represented

Question Q1 (Table 3.A) asked respondents to list their primary project. Table 3.2 provides the results. The number in parenthesis represents the number of respondents who listed that project. As an indication of the frequency of code review in these projects, approximately 83% of the OSS respondents and 90% Microsoft respondents indicated that more than 75% of the code changes in their projects undergo code review. Furthermore, almost two-thirds of the respondents in both surveys indicated that they submit every code change for code reviews. Therefore, the survey respondents represent projects that are actively using contemporary code review.

Table 3.3: Demographics of the respondents

Question	Mean		Median		Category description	% of Respondents	
	OSS	Microsoft	OSS	Microsoft		OSS	Microsoft
Q2. Experience in software development	7 years	10.7 years	5 years	9 years	<i>Low:</i> Less than 2 years <i>Medium:</i> 3 to 5 years <i>High:</i> 6 to 10 years <i>Veteran:</i> More than 10 years	20% 33% 26% 21%	8% 19% 33% 40%
Q8. Average number of hours per week spent in reviewing other contributors' code	6.4 hours	4.7 hours	5 hours	4 hours	<i>Low:</i> Less than 2 hours <i>Medium:</i> 3 to 5 hours <i>High:</i> 6 to 10 hours <i>Very High:</i> More than 10 hours	30% 32% 26% 12%	26% 48% 21% 5%
Q9. Number of contributors' code reviewed each week	6.3 peers	5.5 peers	5 peers	5 peers	<i>Small:</i> Less than 2 peers <i>Medium:</i> 3 to 5 peers <i>High:</i> 6 to 10 peers <i>Very High:</i> More than 10 peers	20% 45% 27% 8%	12% 58% 25% 5%

3.4.2 Respondent demographics

Table 3.3 shows the results from survey questions 2, 8 and 9. In each case, we grouped the responses into four categories by analyzing the frequency distributions of the responses. We then checked these categories to ensure they also made logical sense.

For the OSS survey, 60% of the respondents were paid to work on the project and 40% were volunteers (Q5). The percentage of paid participants is not surprising because the list of projects we drew from included some large, sponsored projects (e.g. Android, Chromium OS, Qt project, and OpenStack), in which most of the participants are employees of the sponsoring companies. This distribution is slightly higher, but in the same range as previous OSS surveys that had 40%-50% paid OSS participants (Bosu et al., 2014; Lakhani et al., 2002).

For the Microsoft survey, approximately 69% of the respondents indicated that most of the code review requests come from developers at the same site, 13% come from a different site, and the remainder are equally split.

3.5 Results

The following subsections describe the results of the two surveys for each of the seven research questions introduced in Section 3.2. To help clarify the results, we also include excerpts from the qualitative responses to the open-ended questions. In this section, we identify each of the respondents using a unique identifier, with *OSS-XXX* and *MS-XXX* indicating respondents from the OSS Survey and the Microsoft Survey, respectively. Unless explicitly stated, the opinions of the OSS and Microsoft respondents were similar. Therefore, the chosen quotations best

represent the set of responses from both samples (OSS and Microsoft).

As a result of the coding process (Section 3.3.5.2), each of the open-ended questions had a large number of detailed categories. For this presentation of the results, we abstracted the detailed categories into a smaller number of high-level categories. Further analysis of the data using the more detailed categories can be found on a supplemental website¹⁴. In a qualitative analysis, each open-ended response could match multiple codes. Therefore, the sum of the percentages could be greater than 100%.

3.5.1 RQ1: Importance of code reviews

In response to Q6, 98.6% of the OSS respondents and 100% of the Microsoft respondents considered code reviews to be important for their project. Figure 3.2 shows the reasons why the respondents found code reviews to be important for code quality (Q7). Although, the relative order of the responses was the same in the both surveys, the distribution of answers was significantly different between the OSS respondents and the Microsoft respondents ($\chi^2 = 18.0, df = 5, p = .003$).

The OSS respondents emphasized code quality slightly more than than the Microsoft respondents did. Because OSS participants come from diverse locations, backgrounds and expertise levels, the quality of submitted code can vary greatly. Therefore, OSS contributors have to focus more on maintaining consistent quality during code reviews. Conversely, there are less quality variations in code changes from Microsoft developers. Therefore, Microsoft respondents are able to focus more on finding defects, and improving their project awareness during code reviews.

In discussions with Microsoft developers, they mentioned that knowledge sharing is one of the primary purposes of code review. Newcomers to a team often are included on reviews not necessarily to improve the code, but so that they can learn more about the codebase and can also see how code reviews are conducted. In some cases there is an explicit mentor-mentee relationship between an expert and a less experienced developer and this manifests in code reviews. We are unaware of similar use for code reviews in the OSS space.

Although one would assume the primary motivation for code reviews is elimination of

¹⁴ <http://carver.cs.ua.edu/Data/Journals/CodeReviewSurvey/>

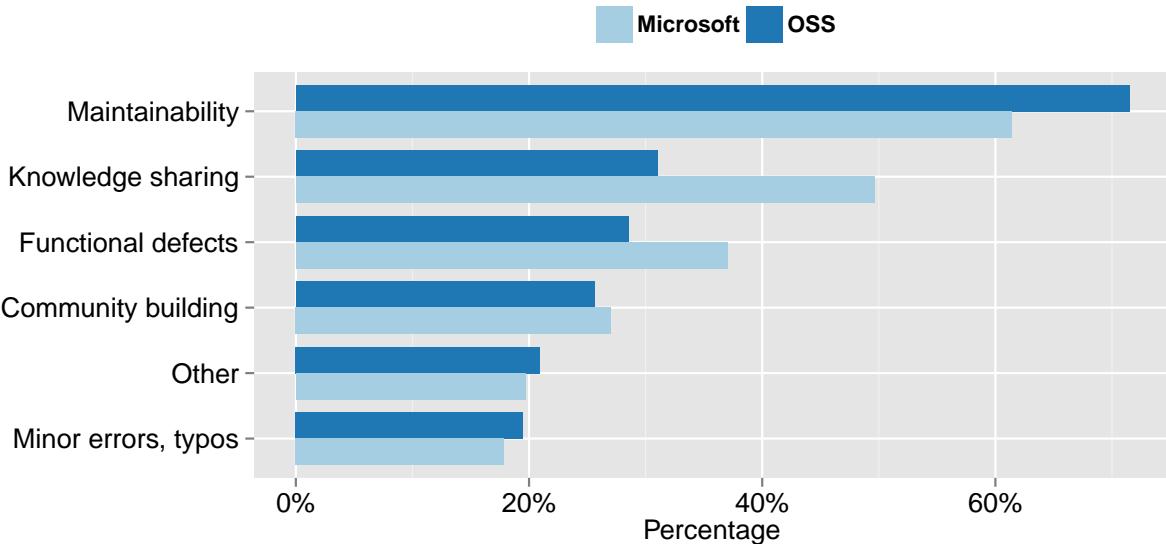


Figure 3.2: Importance of code reviews

defects, *eliminating functional defects* was only the third highest ranked item in both surveys. This result does support earlier findings that the other benefits of contemporary code review i.e. knowledge transfer and identifying better solutions, may be more important than defect detection (Bacchelli and Bird, 2013). Moreover, our prior work also found that approximately half of code review comments relate to maintainability issues and less than a quarter are about functional defects (Bosu et al., 2015). The following subsections provide details on each of these reasons.

3.5.1.1 *Improves Maintainability*

The majority of respondents from both surveys (OSS: 71%, Microsoft: 61%) indicated that code review improves project maintainability in terms of efficiency as well as other maintainability attributes, i.e. legibility, testability, adherence to style guidelines, adherence to application integrity, and conformance to project requirements. In general, developers are more cautious when they know code undergo peer review.

If you know someone is going to look at you, you dress better. When you know someone is going to question you for certain decisions, either you don't make them or you are prepared to defend it. So, in general, it improves quality and makes you better developer by forcing you to look at your code the way others look. [MS-50]

In addition to code quality, reviewers evaluate the code's conformance to project requirements. This conformance is especially important in OSS projects where contributors can have different personal goals.

By requiring approval from core maintainers, it also helps to keep undesirable code out. [OSS-48]

Code reviews and the subsequent discussions also help maintaining project design constraints and result in better designs.

It allows experts on particular areas of the (vast) codebase to detect issues with changes early, helps generate improved design ideas. [OSS-118]

Similarly,

...reconsidered crucial design decisions and ended up thinking “wow, the way I was doing it is stupid for a lot of reasons.” [MS-80]

Another benefit of code review is the production of more readable code. To help reviewers understand the code easily, developers use documentation, comments, and appropriate indentation to make code more readable.

Code review helps to see if “my code” is read and interpreted the way it should be. [MS-25]

Readable code also helps long-term maintainability, especially for large-scale and long-lived projects.

The code is our most valuable asset as well as our biggest liability. But we rarely have the time to re-invest in features done, so it is vital that whatever we checkin is of the right quality. You can test so-and-so much, but you really cannot test maintainability and how easy a codebase is to debug. So to avoid bug-farms you really have to review. [MS-312]

Code review also helps enforce a common coding style, which is one of key characteristics of maintainable code.

Code style is even important because code is written once but read so many times more. In fact, code can be maintained by other devs so it's important it follows guidelines. We have strict coding guidelines published. [MS-290]

3.5.1.2 Facilitates Knowledge Sharing Among Authors and Reviewers

Code review facilitates multiple types of knowledge sharing. Code review interactions help both authors and reviewers learn how to solve problem using new approaches. Reviewers often not only identify issues but also explain why the author's approach could lead to potential problems. Reviews also help socialize project details, e.g. architecture, common APIs, and existing libraries.

... spread information to more people so all knowledge of a system is not lost if someone is out sick, on vacation, or leaves the team, assist in sharing knowledge of helpful utilities so that we do not end up with duplicate systems doing the same things. [MS-196]

Code reviews increase project awareness among the project members by ensuring that at least one or two reviewers are also aware of code changes.

... it helps to ensure that more than one member of a group is familiar with any changes, it makes sure that all changes are (at least somewhat) sane, and it helps to foster feedback from people who will be affected by a change before the change actually happens. [OSS-194]

Code reviews also allow senior project members to mentor newcomers.

... code reviews are often one of the primary methods of knowledge transfer and brainstorming about software between developers. They're part of the critical path to ramp-up new developers on both the project and technologies, and they're often where experienced developers share tricks-of-the-trade and knowledge in context. [MS-190]

In addition to newcomers, more experience project members can also learn through code reviews.

... allows us to leverage the lessons learned by each person in the code base that not everyone will encounter. [MS-355]

3.5.1.3 Eliminates Functional Defects

Reviewers often find logical errors, corner cases, security issues, or general incompatibility problems that the author may have overlooked.

Code review dramatically reduces bug count, in my experience. It is very rare for a change to be accepted without some suggested improvements or notations of deficiencies by reviewers. [OSS-145]

Experienced security reviewers are often able to identify critical security flaws during code reviews.

More people see more, you can not let anyone from the community to merge anything to your code (security risk). [OSS-105]

Finally, code reviews help to inform wider audience about agreed upon changes and thus help avoiding incompatibility issues (i.e., broken build).

... makes sure the feature/bug fixing can integrate into other parts of project, done by other developers. [MS-241]

3.5.1.4 Community Building

By fostering direct collaboration between developers and reviewers, code reviews encourage community building and collaborations. Community building is very important especially in OSS projects.

.. helps the developer feel part of the F/OSS community. It also provides a framework for participation, and allows people to voice opinions on submitted changes and feel involved. [OSS-39]

In addition, the inclusion of various stakeholders in the review process can reduce collaboration issues for large scale projects.

.. we are all (as developers) trying to contribute to a much greater project. By doing a code review, we share our changes with others to help inform them, and in return they can share their wisdom of the project with us about whether our changes are accurate/appropriate. [MS-34]

Finally, code reviews also help developers gain a better perception of each others' expertise and build relationships.

Gives a sense of collaboration and camaraderie among engineers who would not typically work together (employees of competing companies, for instance). [OSS-37]

3.5.1.5 Minor Errors, Typos

Developers often do not notice their own minor errors and typos. Without code reviews, identifying those minor issues may be time-consuming. In addition, developers may forget to keep comments updated, which is crucial for long term maintainability of the project. In most

cases, the majority of the minor errors or typos are identified during code reviews.

It helps catch human errors/typos. Two pairs of eyes are always better than one. [MS-182]

3.5.1.6 Other

A small number of respondent mentioned other important aspects of code reviews such as promoting collective code ownership, multi-vendor involvement, improving the chance of newcomer adoption through mentoring, and avoiding conflicting changes, building a shared vision, and leveraging more eyeballs.

3.5.2 RQ2: Time spent in code reviews

According to Q8, the average time spent in code review each week is 6.4 hours for OSS developers and 4.7 hours for Microsoft developers. Considering 45 work-hours per week, this result indicates that *developers spend 10% - 15% of their time in code reviews*.

Because a less experience developer would be more likely to invite an experienced teammate to perform a code review, we hypothesize that experienced developers would spend more time performing code reviews. Figure 3.3 shows development experience category (Table 3.3) vs. mean hours spent in code review. Since the distribution of mean review hours significantly differs from a normal distribution, we used non-parametric ANOVA (i.e., Kruskal Wallis H), which indicates that those differences are statistically significant (OSS: $\chi^2 = 8.16, p = 0.043$, Microsoft: $\chi^2 = 8.43, p = 0.038$). The results support the hypothesis that hours spent in code review usually increases with development experience.

For the OSS respondents, the paid contributors spend significantly (Mann-Whitney U, $p < .001$) more time in code review than volunteer participants, median of 5 hours vs. 3 hours . This results makes sense because paid contributors act as gatekeepers to maintain the integrity of the software by preventing buggy, unwanted or malicious code. As a gatekeeper, the paid participant will therefore review code from many different peers and spend more time in code reviews. To support this observations, the results of Q9 indicate that paid contributors review code from significantly (Mann-Whitney U, $p=.009$) more peers each week than volunteers do, median of 5 peers vs. 4 peers.

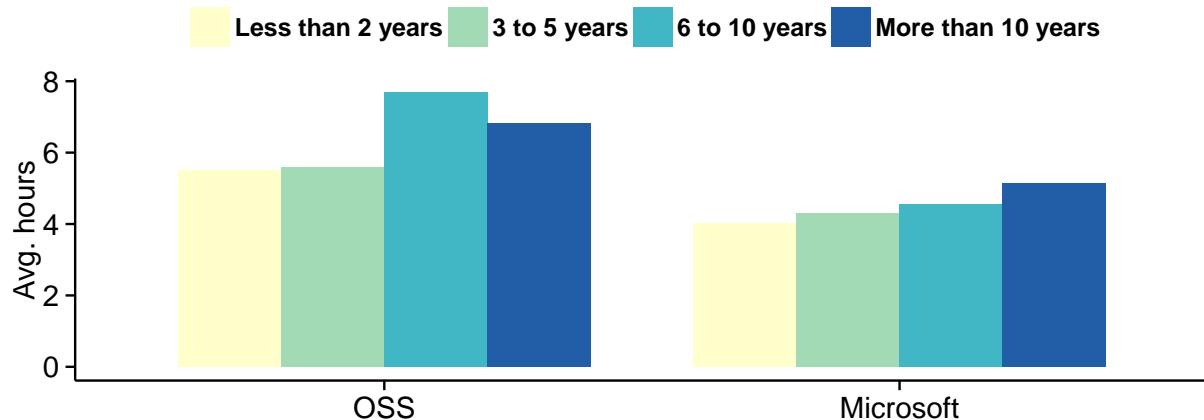


Figure 3.3: Hours spent in code reviews vs. Experience

3.5.3 RQ3: Accepting review request

More than half of the OSS respondents and two-thirds of the Microsoft respondents indicated that the identity of the author was important in accepting a code review request (Q11). Figure 3.4 shows the reasons why respondents found the code author's identity important (Q7). Although the factors identified were common between the two surveys, the distribution of responses was significantly different ($\chi^2 = 69.5, df = 4, p < .001$). For example, the non-technical factors (i.e., reputation and relationship) were the most important for the OSS respondents, while they were the least important for the Microsoft respondents. This result reinforces the emphasis that OSS developer place on reputation and relationships found in other research (Raymond, 1999; Lakhani and Wolf, 2003).

Conversely, the expertise of the contributor and time / effort required for the review were the primary considerations for the Microsoft respondents. Discussions with developers shed some light on the reasons for this result. With respect to expertise, Experts in a mentoring relationship will often select to review code from low expertise developers while changes from more skilled developers may not need as heavy scrutiny. In addition, since developers at Microsoft must manage competing demands for their time and products have tight timelines, developers must frequently make decisions based on the time required to complete a task. Thus, the choice to participate in a code review depends heavily on the estimated time required. The following subsections provide details on each factor.

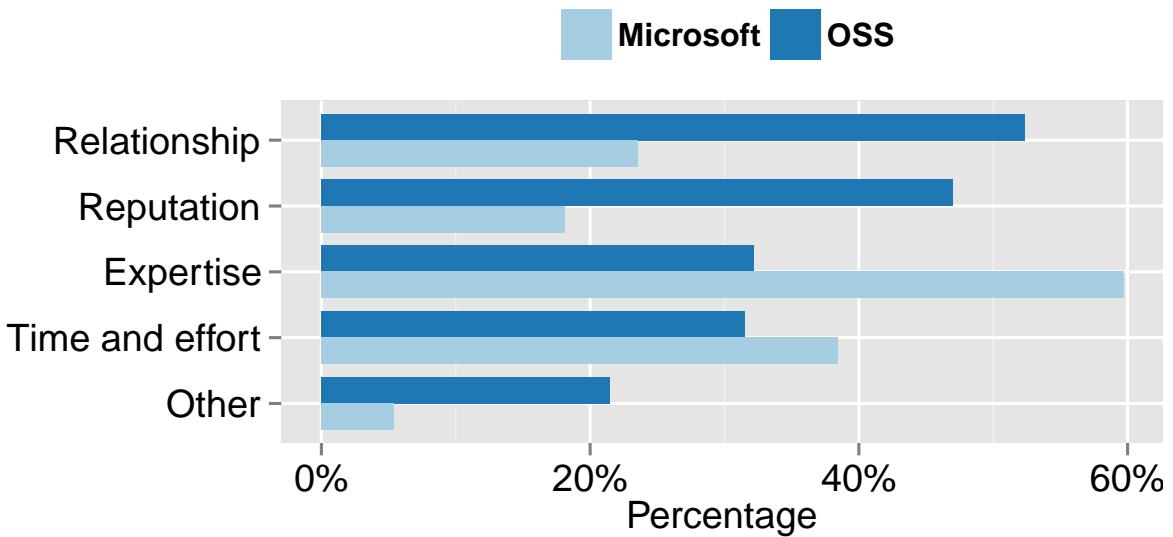


Figure 3.4: Why the identity of a code author is relevant

3.5.3.1 Relationship

A reviewer's relationship and history of interaction with the code author often affects the decision of whether to accept a review request.

We know each other. We know each others strengths and weaknesses and we can change the way we review to meet to needs of the specific developer. It is an optimization that humans naturally perform. Is it a net positive? I think so. [OSS-60]

Furthermore, to optimize the time spent reviewing code, a reviewer often chooses to review code from authors who have already reviewed his/her code.

It feels like a “quid pro quo” - if the contributor has reviewed my code in the past in a thorough/timely fashion, I like to return the favor. [OSS-20]

Next, because code changes from a trustworthy author are more likely to require less reviewing effort, the level of trust between the reviewer and the author is important.

If you review code from someone you already know and trust very well, you can only focus more on detecting careless mistakes and less on overall design of the code change. [OSS-276]

Finally, reviewers prioritize requests from their teammates or co-workers over others.

... there are other programmers at my company that also work on the project, and if they've submitted something that is time-sensitive, I'm likely to prioritize that review to keep things

moving. First-time contributors tend to get down-prioritized a bit. [OSS-227]

3.5.3.2 Reputation

Depending upon their goals or project roles, some reviewers may seek out authors with positive reputations, while others may focus on those with poorer reputations. To leverage the time spent in code reviews, some reviewers favor review requests from authors they consider to be as capable of producing high-quality code.

Often time to do reviews is limited. I prefer changes from contributors where I know that they are proposing good changes (high code quality, good commit message, small scope, focused on one thing), because I know that I can finish the review quickly. I also prefer changes from contributors that themselves give feedback by doing code review on changes of others.

[OSS-106]

Conversely, some reviewers are gatekeepers that focus on code changes from new or troublesome authors.

I am more likely to review changes by developers new to the team, as well as developers who have a history of poor adherence to coding standards or lots of significant comments on their reviews. I am also more likely to review a change if a developer is making a change outside of their typical area of expertise (for example, a backend developer making a UI change, or vice versa). [MS-90]

3.5.3.3 Expertise / Experience

For the Microsoft respondents, an author's experience or expertise was the most important factor, by a large margin. Many reviewers prefer reviewing code changes that are closely related to their areas of work or expertise.

I get a lot of code review requests from multiple teams. I only review things that are in my area, and author of change often helps to determine if changes are relevant to me. [MS-325]

Sometime the expertise of the code author also influences the decision to accept or reject a review request. Developers will often accept review requests from experienced contributors expecting to learn about outstanding techniques or designs. Conversely, experts and/or code owners may be more likely to review code coming from inexperienced developers in an effort to maintain high code quality.

Some people do stellar work, and I want to learn from them, so I review their CR to see what they did, even though I almost never find problems. [MS-319]

3.5.3.4 Time / Effort to Review

Based on the author's identity, reviewers can often anticipate the amount of effort required. To maximize the utility of time spent in code reviews, some reviewers focus on areas that require the most attention.

My time is inherently limited, so I choose to prioritize code review for less experienced developers. For code from people that I know have a history of quality contributions, I'm less likely to spend time reviewing.[OSS-176]

Conversely, to reduce their effort, some reviewers prefer to avoid changes from known poor coders.

Sometimes you want to quickly review code from contributors whose work you trust greatly. Other times you might choose to ignore work from a known contributor who typically produces poor work. [OSS-114]

3.5.3.5 Other

Because encouraging participation of new contributions is important for the health of OSS projects, reviewers sometimes accept requests from newcomers specifically to mentor them and encourage their contributions.

Fostering new people in participating is something we try to actively do in the project. [OSS-76]

Finally, a few reviewers chose reviews based on their interest in learning something new.

It's not the identity of author per se but more whether author works in the field/project I'm interested in. For learning purposes I sometimes review changes from some authors even if project is quite far from me. [MS-164]

3.5.4 RQ4: Characteristics of low quality code

From our previous work (Bosu et al., 2014) and common code smells (Fowler, 1999) we identified ten characteristics of low-quality code. In the OSS survey, we asked the respondents to rank those characteristics based on their importance during code reviews. However, due to

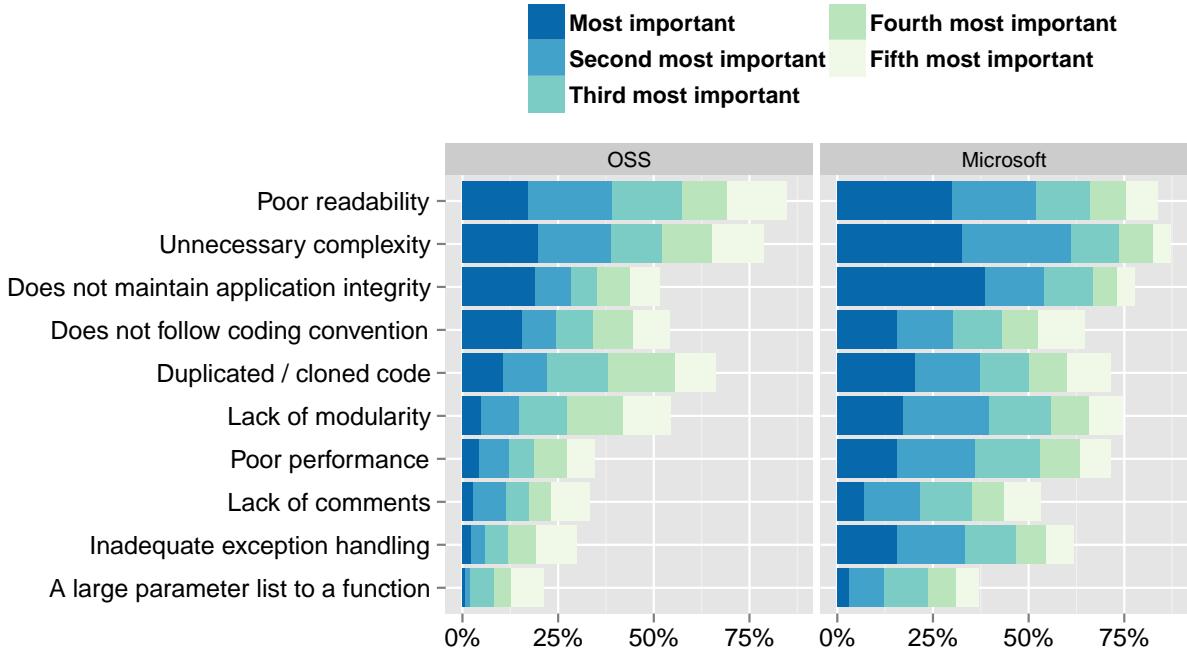


Figure 3.5: Characteristics indicating poor code (Sorted based on the ranks in the OSS survey)

the limitations of the survey tools, in the Microsoft survey, respondents rated each characteristic on a 6-point scale rather than rank ordering them. Note that the OSS respondents could rate only one characteristic as being *most important*, while Microsoft respondents could rate multiple characteristics as *most important*. As a result, the total for *most important* is greater than 100% for the Microsoft survey. However, we believe that this may not be an issue, since we are interested only in comparing the ranks of the characteristics between the two surveys. For the two surveys, we separately calculated the ranks of the characteristics using the number of top two ratings (i.e. Most important, and Second most important). Figure 3.5 summarizes how the respondents rated the relative importance of each characteristic (Q17), where the characteristics are sorted based on their ranks in the OSS survey.

The fact that *unnecessary complexity* and *poor readability* were among the top three characteristics in each survey, suggests that code which is simple (not complex) and readable is easier to review. It is interesting to note that *lack of comments* ranked very low (OSS: 8th and Microsoft: 9th) in both of the surveys. The combinations of these results suggest that reviewers expect code to be straightforward and self-documenting rather than requiring extensive comments to explain it. Because a reviewer has to understand the code to properly review it, a

complex approach, even if well-commented, will likely take longer to review.

Does not maintain application integrity was also among the top three characteristics in both surveys. For long-term project maintaining a consistent design is very important. A feature that violates project design not only adds burden for future maintenance but also opens bugs or even vulnerabilities. Such code generally indicates either the author lacks knowledge about the project design, or the author lacks care / dedication for the project. Therefore, authors should be careful that submitted code changes maintain application design constraints.

The ranking of eight of the nine characteristics was similar for both surveys (with a difference of no more than two ranks). The exception was the characteristic: *does not follow coding convention of the project* (fourth in OSS and eighth in Microsoft). There are two possible explanations for this result. First, while Microsoft respondents may consider coding convention issues important, they may not judge code quality based those problems because they are easier to fix. Second, because Microsoft developers often use automated tools to identify and fix coding convention issues, they may focus less on these issues during code review.

3.5.5 RQ5: Assistance to improve low quality code

Figure 3.6 lists the approaches the respondents used to help poorly written code reach the level of quality required for inclusion in the project (Q15). The distribution of responses was significantly different between the OSS respondents and the Microsoft respondents ($\chi^2 = 105.2, df = 6, p < .001$). This difference was largely due to two factors.

First, the Microsoft respondents are more likely to communicate with the author using other channels (i.e., face-to-face, Skype, instant messenger or email). They found those communications helpful in quickly resolving any misunderstandings. Conversely, face-to-face communication may not be an option in an OSS project. Interestingly, OSS developers could use some of the tools (e.g., Skype or other voice/video over internet technologies), but they do not.

Second, when other methods are unsuccessful, the OSS respondents are more likely to rewrite the code themselves. Because OSS participants may not be obliged to follow up, they may not make the changes required to make the code acceptable. If a code change is important, then the reviewer, may choose to just fix the problem rather than waiting on the original author. Conversely, the Microsoft respondents rarely rewrite poor code themselves, for two primary

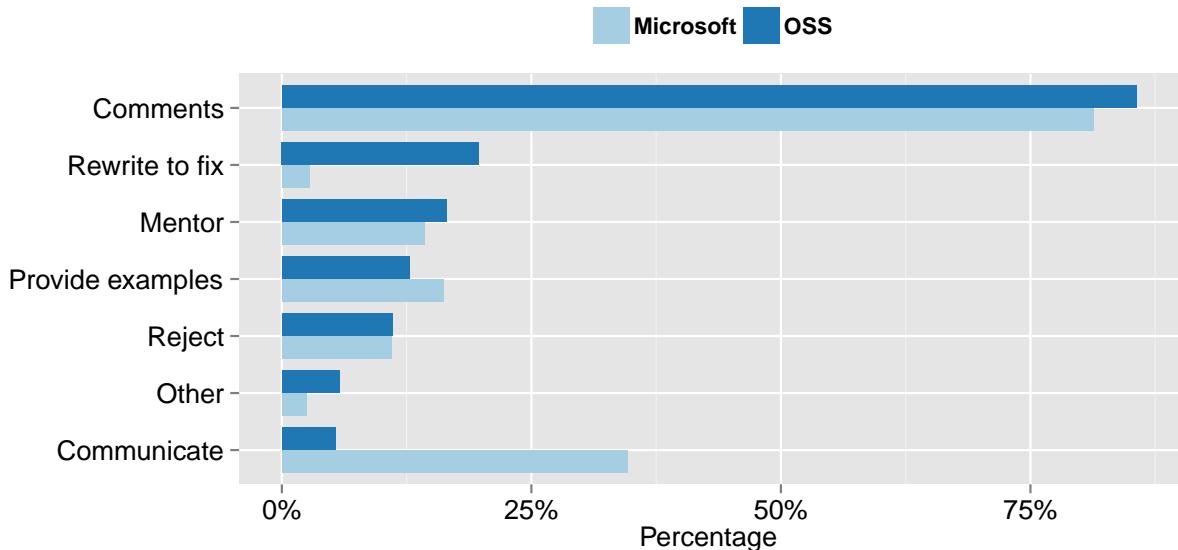


Figure 3.6: How reviewers assist to fix a poor code

reasons: 1) Microsoft developers are required to follow up, and 2) reviewers know that they have to mentor authors of low quality code to help them learn how to write better code.

3.5.5.1 *Provide Comments*

More than 80% of the respondents from each survey provide comments through the code review tool to help authors improve poorly-written code. The reviewers typically indicate specific shortcomings of the code and ask the author to fix those issues.

For issues specific to the patch in question, or small coding style/convention issues, I'll reply to the patch with point-by-point feedback and suggestions. For major systemic issues, such as pervasive use of incorrect coding style/conventions, or fundamental architectural issues, I'll reply to the first instance of such an issue with a summary of the problems and an indication that many more exist that I didn't quote or comment on. [OSS-46]

Many reviewers provide hints or suggestions to refactor the code to make it more readable.

I try and give syntax tips or suggest improvements (like rewriting a function to reduce complexity, pointing out where we have duplicate code and how it might be shared and suggesting to split a large function into smaller ones). [MS-403]

A few respondents also mentioned the importance of constructive criticism to avoid hurting the feelings of the code author.

Critique the code, not the author. Describe better approaches, don't just denigrate the chosen

one. Ask questions about why an approach was chosen, don't attack the choice. Point out possible edge cases where they are overlooked. Refer directly to the coding standard where appropriate. [MS-73]

3.5.5.2 I Rewrite/fix the code

In projects where authors are not required to respond to reviews, i.e. some OSS projects, the some reviewers find it quicker to just fix the low-quality code rather than providing comments.

... in some circumstances I massage the code change myself and explain to the submitter why I have made the follow-up change. [OSS-276]

Although, the reviewers do realize this practice may not be the best approach.

Usually it boils down to rejecting it or fixing it by writing it myself. I am aware that this is not a good practice, but we're all volunteers. [OSS-154]

3.5.5.3 Provide Mentoring

In cases where the author of low-quality code lacks project or programming knowledge, mentoring may be the best approach to improve code quality.

I try to provide design guidance to the contributor when I think it will benefit the project. I may also provide some language-specific mentoring or at least refer the contributor to relevant documentation I believe to be helpful. [OSS-71]

Another type of mentoring is to ask questions to help the author understand potential code problems.

Ask questions such as what happens in scenarios to guide him through this understanding. If this is due to lack of understanding of fundamental technology then give pointers to bring up the knowledge. [MS-126]

3.5.5.4 Provide examples

Some reviewers prefer providing example code or directing an author to other well-written code in the project.

I will usually point the original author towards existing examples of code in the project to look at for reference. [OSS-58]

3.5.5.5 Reject until good

Some reviewers prefer to reject code changes until they meet the project quality standards.

I do not sign off until I am convinced that the code change meets the team criteria for quality. If the author does not understand or agree with my feedback, I typically will sit down with them to discuss in detail. [MS-132]

3.5.5.6 Communicate with the author

Over a quarter of the Microsoft respondents preferred discussing code changes via email or instant messenger rather than inside the code review tool. They found this type of communications helpful for avoiding long discussions in the code review tool and embarrassment of the author.

Sometimes, for more difficult issues, I create an email thread on the side to have a better back-and-forth discussion about the change as a whole instead of a discussion about one small part indicated in the review. If I consider the change very poorly written, I tend to keep the side-conversation more private to avoid embarrassing the author. [MS-145]

If feasible, some of the Microsoft reviewers also prefer to meet the author face-to-face and discuss about the issue to resolve potential misunderstandings.

Usually the best option is to go to their office and see where they are coming from and whether they made oversights or were missing information. [MS-34]

3.5.5.7 Other

Some of the respondents indicated that they type of feedback they provide for poor code depend on the identity of the code author.

If the developer is a beginner I help it to improve his code, give advices, ... If the dev is supposed to be “good”, I just put a -2. [OSS-221]

Other strategies to help poor quality code include: flagging poor code changes, asking the author to write unitests, and suggesting the author to recompile the code with warning flags enabled.

3.5.6 RQ6: Impact of high quality code

More than 85% of the respondents from each survey indicated that high quality code or use of an outstanding approach to solve a problem affects their perception of the code author (Q17). As shown in Figure 3.7, the aspects of peer perceptions that are influenced by high quality code (Q18) differ significantly between OSS respondents and the Microsoft respondents ($\chi^2 = 46.9, df = 3, p < .001$).

For the OSS respondents, the largest impact of high quality code is increase in positive impressions about the personal characteristics of the code author. Because of the lack of physical interaction among OSS participants, socio-technical interactions (e.g., via code reviews) become more influential in the formation of impressions about the personal characteristics of teammates (Bosu et al., 2014). On the other hand, the lower proportion from Microsoft respondents may be a reflection of the many other ways that developers can assess the characteristics as they interact and observe each other; they will often have meetings together, work in close proximity, coordinate frequently, and participate in non-work activities (e.g. going to lunch together), all of which in general is more prevalent in an industrial than an OSS setting.

Conversely, the Microsoft respondents indicated that the largest impact of high quality code is stronger relationships and future collaborations with the code authors. Because approximately 75% of the code reviews at Microsoft are performed by teammates of the code author (Bosu et al., 2015), the reviewers are likely already aware of the personal characteristics of an author. Instead, code reviews help the reviewers judge the intellect and coding skill of the code author. High quality code can lead to increased respect, admiration and trust.

3.5.6.1 Personal Characteristics

Approximately 60% of the OSS respondents and 36% of the Microsoft respondents indicated that high quality code or an outstanding problem solving approach were evidences of the personal characteristics of the code author. First, high quality code can indicate that the author is competent, an expert in the area, high performing, and posses professional skills.

I will assume that the author is experienced or skilled in a particular area if I see good work in that area. If the author tends to produce clean, well structured code that complies with the coding style guide, I will assume that he/she works in well organized and thorough manner.

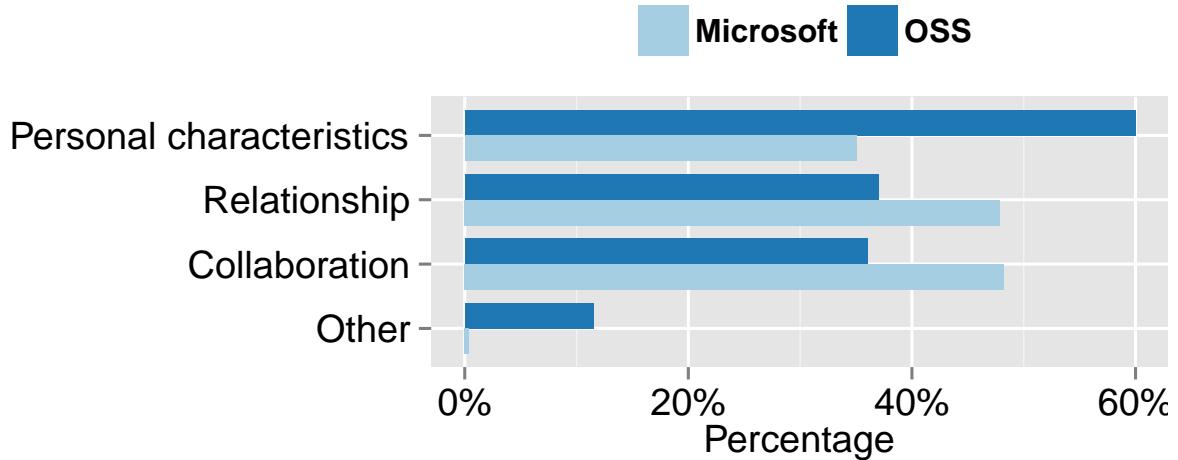


Figure 3.7: Impact of a high quality code

[MS-68]

Second, high quality code can be a sign of the contributor's ability.

It's my opinion that you can infer the quality of a developer by the quality of their changes.

Changes that are not well thought out indicate sloppy thinking, but changes that are neat and tight indicate an accuracy of thought that I appreciate. [MS-216]

Third, the quality of code changes can indicate the level of the dedication of the code author.

High quality code shows that the author cares about the project and has considered the ramifications of their changes. High quality code also can elevate the project as new ideas are injected into the community. [OSS-31]

Finally, high quality code can be a sign that the author has a good understanding of the project.

This means the author has spent a large amount of time working hard to understand the problem at hand and has come up with a great solution. [OSS-220]

3.5.6.2 Relationship

Approximately 37% of the OSS respondents and 47% of the Microsoft respondents indicated their desires to build relationships with outstanding code authors. They believed that authors of good code are trustworthy and should have additional tasks and privileges. Trust is very important in OSS projects, because gaining the trust of the core members is the only way a contributor can earn commit privileges.

.. “outstanding approaches” are rare, but well written, well-documented code indicates

that the author can be trusted. He/She may get approver rights or become maintainer of a module. [OSS-167]

In addition, respondents reported an increase in respect and admiration for authors of high quality code.

High quality code speaks about its author. I see software development more as an art than an actual engineering discipline. From this perspective, a person writing high quality code is someone that deserves respect and recognition, as he combines his knowledge/experience with his intellect to create unique solutions. [OSS-180]

Finally, authoring high quality code can also lead the author to earning a better reputation within the community.

Impressed by ability to solve the problem, this is really just meritocracy at work. People who put in the time and solve problems with outstanding approaches build a strong reputation. [OSS-144]

3.5.6.3 Future Collaboration

Approximately 36% of the OSS respondents and 48% of the Microsoft respondents indicated their desire for future collaborations with authors of high quality code because they viewed those authors as expert contributors from whom they could learn.

If someone writes some code with an “Outstanding approach” such that it impresses me, I’ll probably read all of their code reviews after that, in order to learn more. [MS-103]

In fact, some developers often volunteer to review code changes submitted by these authors primarily to learn.

I will be more likely to consult this person in the future as they are a proven performer in code matters. I may pay more attention to future work by them by signing up for code reviews, but more to understand their work than to pick apart code line by line. [MS-333]

Apart from learning, developers also seek assistance from outstanding code authors when having difficulty solving a problem.

I know that this is a guy to go to when faced with difficult problems, and that he can be counted on to give proper reviews and suggest improvements to my code. [OSS-109]

Consistently submitting high quality code can improve the trustworthiness of the author as respondents stated that, if busy, they would spend less time reviewing code from these authors.

.. if this person sends another code review on a day that I'm really busy, I won't worry about looking it because I trust that they also "did the right thing" in this new code review.
[MS-128]

Finally, authoring high quality code changes resulted in an increased perception of reliability and the assignment of more complex or critical.

Seeing well written code increases my confidence in the author and I know I will be able to rely on that author for future tasks of high complexity or high importance. [MS-341]

3.5.6.4 Other

In popular OSS projects, a contributor can become a core member only after providing high quality contributions over a period of time. A few OSS respondents indicated that the submission of high quality code changes is often an early indication that author may become a maintainer or core member.

Usually indicates understanding of the background of the project and understand our common goals. for a first time contributor, it is immense respect, especially if followed through. for a repeat contributor, it goes to show he could be a future maintainer as he is in sync with community needs. [OSS-211]

3.5.7 RQ7: Impact of low quality code

More than three-quarters of the respondents from each survey indicated that poorly written code negatively influences their perceptions of the code author (Q13). As shown in Figure 3.8, the aspects of peer perception that are influenced by poorly written code (Q14) differ significantly between OSS respondents and Microsoft respondents ($\chi^2 = 60.3, df = 3, p < .001$). The OSS respondents were more likely to form negative impressions about the experience and personal work habits of authors of poorly written code. The reasons for forming negative impressions are largely the same as those for forming positive impressions (RQ6). In addition, the Microsoft respondents considered authors of low quality code to be incompetent and were less likely to collaborate with those authors in the future due to the expected increase in time and effort such collaborations would require.

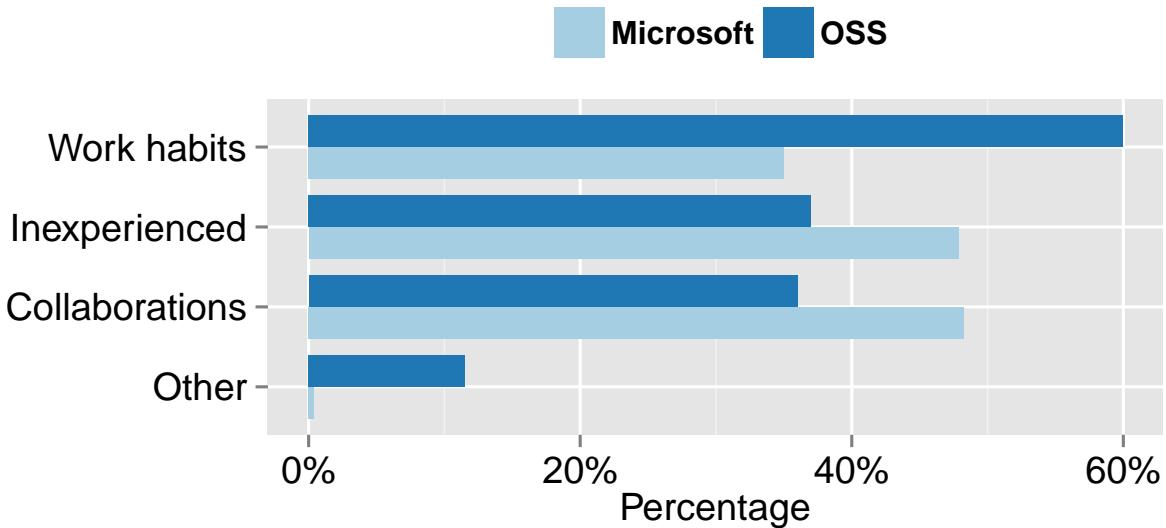


Figure 3.8: Impact of a poor quality code

3.5.7.1 Work Habits

Approximately 62% of the OSS respondents and 38% of the Microsoft respondents indicated that low quality code negatively affects their perceptions about the work habits of the author. As before, Microsoft respondents likely find this factor less important than OSS respondents because of the many other ways that developers interact and are able to assess each other's characteristics and work habits. More specifically, reviewers consider the submission of low quality code to be a sign of the author's carelessness or lack of respect for his/her peers.

Code that is poorly written, does not follow guidelines etc. makes a bad impression, it gives a sloppy impression, like the person writing the code does not take the time to provide the code with proper quality, yet expect it to be reviewed. Providing sloppy code is in my opinion a sign of lack of respect. [OSS-260]

Second, reviewers think that authors of low quality code were lazy or lacking dedication to the project.

Author either didn't put enough time to investigate the task he is solving or does not have understanding of the system he is building. Either case he should have spent more time to understand what is he doing. [MS-41]

While low quality code does affect perception, reviewers realize that people do make mistakes, so they are likely to excuse the first few mistakes and begin forming a negative impression when

the author is unable or unwilling to learn from previous mistakes.

For occasionally poor code, not much effect - people have off days, or may just misunderstand the particular area of code they are modifying. But if the author continues to makes similar mistakes, then that strongly degrades my opinion of their competence. [OSS-176]

Along these same lines, not all the mistakes have the same impact. Reviewers are more accepting of mistakes due to lack of knowledge or understandings than they are to easily avoidable mistakes.

It's ok if the code contains bugs and I will not think the author is careless. However if the code contains coding style, readability, duplicated code and other easily avoidable problems, I will think the author is careless and is not dedicated to the project. [OSS-197]

3.5.7.2 Inexperienced contributor

Approximately 51% of the OSS respondents and 24% of the Microsoft respondents indicated that low quality code suggests that the author is inexperienced in either the project or in programming.

It can mean that they haven't yet reached understanding of the how the code they are modifying works, and thus any contributions from them may need to be reviewed extremely carefully. [OSS-196]

Reviewers also think that the authors of low quality code may be incompetent or lacking intelligence.

No matter what level of experience a programmer is at they can write clear, readable, robust code. Surprisingly people can go a long ways without learning to do this. It makes me question their native intelligence and dedication to the project. [OSS-25]

3.5.7.3 Future collaborations

Approximately 62% of the Microsoft respondents and 27% of the OSS respondents indicated that the impressions formed about authors of poorly written code affected their future collaborations. Many of the respondents doubted a poor code author's abilities.

Poorly written code usually indicates to me that the author is lacking coding experience or technical skill, which (negatively) affects how I perceive the author's general performance at work. [MS-45]

Another factor hampering future collaboration is loss of respect.

Depending on the ‘severity’ of the bad code, I can feel that I lose respect for the person and their intelligence as a code author in extreme cases. For example thinking ‘did they even try to build it/test it’, or ‘why did they think this is the right approach. It should be much simpler but they are too clueless to know’. [OSS-78]

In the extreme case, reviewers can lose trust in the author of low quality code and carefully examine future changes from that author.

I trust the developer less, and know that I’ll have to code review future changes in even greater detail. [MS-349]

Finally, because poorly written code takes longer to review, some reviewers are less likely to accept review requests from these authors.

Poorly written changes also require more review time, so I feel that a person who consistently makes poorly written changes will waste a lot of people’s time in the long run. I also end up expecting that person’s changes to be poorly written and do not look forward to the prospect of reviewing the changes. [MS-166]

3.5.7.4 Other

Some of the respondents mentioned that the impact of a poor code would depend on how the author responded to review suggestions.

This depends on how the code author responds to the comments. If they learn from the comments and feedback provided, then I have a very positive perception. If they argue that “style”, or “compiler warning”, or “documentation” is not important and that we should let the patch proceed as is, then I have a negative perception. [OSS-234]

3.5.8 RQ8: Effect on peer impressions

Using behavioral scales, we focused on understanding impact of code reviews on four aspects of peer impression formation: trust, reliability, perception of expertise, and friendship. To ease analysis and presentation of results, we recoded the scale to make the effect of the scale items on impression formation more evident. The recoded scale is: -3: *describes a non-code review partner, NOT a code review partner*, 0: *describes both equally*, and 3: *describes a code review partner, NOT a non-code review partner*. To avoid biasing the results with negative scale

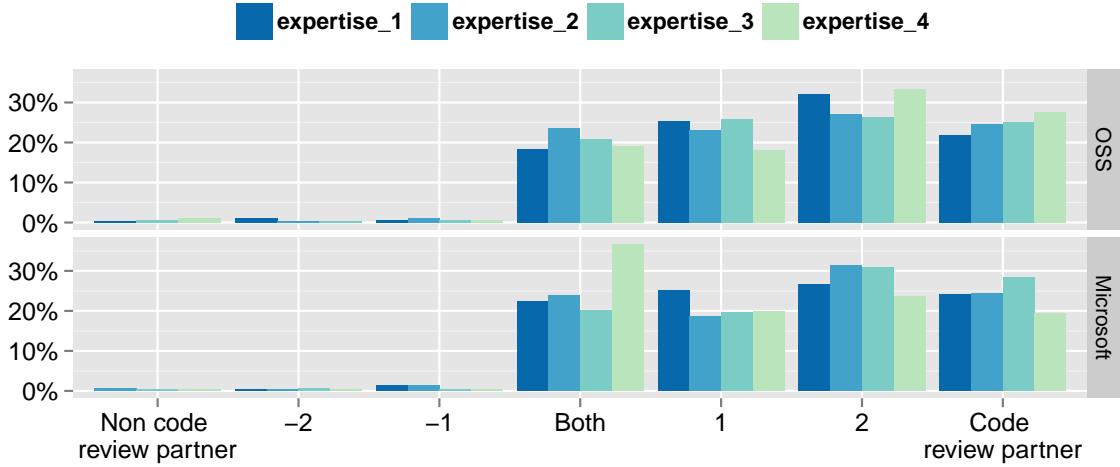


Figure 3.9: istribution of ratings for the scale items: Perception of expertise

values, we did not use this scale during data collection.

As an example, Figure 3.9 shows that for the four perception of expertise scale items, most of the respondents (approximately 70% to 80%) thought they had a better perception of the expertise of their *code review partners* than their *non-code review partners*. All four scales exhibited a similar trend.

Table 3.4 shows the item means for the four behavioral scale items for the two surveys. The scale means were positive and significantly higher¹⁵ than the mid-point of the scale (0 - Both Equally) in all cases. We also estimated effect size using Cohen's *d* (Table 3.4-rightmost two columns). In seven out of the eight cases there were large effects. Only the trust scale in OSS survey showed medium effect size. The results suggest that code review had overall large positive impact on building four forms of peer impressions (i.e., trust, perception of expertise, friendship, and reliability) between code review participants.

These results provide some insight into the results from RQ6 (impact of high quality code) and RQ7 (impact of poor code), which show that code reviews can have both positive and negative impacts on impression formation. This analysis shows that 1) code reviews have a large positive impact on impressions formation, and 2) the majority of the respondents had better perceptions of their code review partners than their non-code review partners.

¹⁵ one sample t-test, $p < 0.001$ in all cases

Table 3.4: Behavioral scale means and effects

Construct	Scale Mean		Effect Size (Cohen's d^*)	
	OSS	Microsoft	OSS	Microsoft
Trust	0.699	0.843	0.76	0.93
Perception of Expertise	1.526	1.467	1.72	1.53
Reliability	1.023	1.088	1.11	1.04
Friendship	1.473	1.115	1.53	1.33

*Cohen's d interpretation: $d \geq .8$ indicating large effect, $d \geq .5$ indicating medium effect, and $d \geq .2$ indicating small effect (Cohen, 1988)

3.6 Discussion

This section provides further discussion on the detailed analysis of the survey results described in Section 3.5. In particular, this section highlights six themes that emerged from the results.

3.6.1 Benefits of Code Review

While there is empirical evidence that code review improves software quality (Wiegers, 2002; Cohen et al., 2006), the benefits of code review are much broader. Evidence about these other benefits has been mostly anecdotal. The results of these surveys begin to provide more evidence for these benefits. Nearly all survey respondents in both surveys found code reviews important for their projects for reasons including: knowledge dissemination, relationship building, better designs, and ensuring maintainable code. For a large scale and or long term projects, those benefits may be very important and hard to achieve through other ways. Therefore, even projects with highly-skilled developers who rarely commit low-quality changes can still benefit from the practice of code review.

3.6.2 Peer Impression Formation

One of the key non-technical benefits of code review is its role in impression formation, that is how developers form opinions of teammates. The results of the surveys show that the most important social factor of code reviews is in obtaining an accurate perception of the expertise of teammates. The quality of the code submitted for review is an important aspect of the formation of teammate perceptions. For example a code change that is simple, easy to understand, self

documenting and requires minimum review time is highly appreciated by the reviewers and may lead to improved social status. Whether a developer is positive or negative towards a teammate may influence future code reviews. For example, respondents indicated that they perform more thorough reviews of code submitted by teammates who are untrustworthy than of code submitted by teammates they view as experts. In addition, the impressions formed during code review could also affect future collaborations, relationships, and work practices. Therefore, code review is a critical practice not only for ensuring the quality of code changes but also for forming the social underpinning of successful projects.

3.6.3 Effects on Future Collaborations

More than three-fourth of the code review participants had strong positive impressions about their peers, suggesting that code reviews may influence future collaborations. When a reviewer finds high quality code from an author it can increase collaborations in two ways. First, s/he is more likely to signup to review the author's future changes in order to learn from them. Second, s/he considers the author an expert who is able to provide suggestions to improve his/her code and add the author as a reviewer for his/her changes. On the contrary, a poorly written code often takes more efforts to review and a reviewer may not accept future review requests from a poor author. Moreover, a developer may not ask a poor author to review his/her code as s/he doubts the poor author's expertise. Combining these two scenarios, a developer's code review partners (i.e., a person who reviews your code or whose code you review on a regular basis) are more likely those peers who s/he considers as good authors as well as experts. On the other hand, the peers who s/he judged as poor code authors will become non-code review partner (i.e., a person who has been a peer for some time, but you have rarely reviewed their code) due to infrequent interactions.

3.6.4 Effects of Distributed vs. Co-located Teams

One of the goals of replicating the survey within Microsoft was to investigate how much the distributed nature of OSS projects factored in to the understanding and practice of code review. We anticipated that members of distributed teams would emphasize the human relationship aspects of code review due to their limited ability to form these relationships in person (as members of co-located teams have). Interestingly, when analyzing the data from the Microsoft

survey we found very little difference between the responses of developers from co-located teams and the responses of developers from distributed teams. In fact, there were much larger differences between respondents to the OSS survey and respondents to the Microsoft survey (regardless of the type of team of the respondent), as discussed in the next section. These results suggest two possible explanations: 1) some types of impressions (e.g. coding ability) that are impacted by code reviews may not depend on face-to-face interactions, and 2) the code review process (e.g., review acceptance and future collaborations) may depend more on project governance style than on the physical location of the developers.

3.6.5 Differences Between OSS and Microsoft

The OSS respondents differ significantly from the Microsoft respondents in the aspects of code review emphasized as most important. The focus for OSS reviewers is on building relationships with core team members. When forming impressions of their teammates through code reviews, the OSS respondents indicated that the personal characteristics and work habits of the code author were most important. This emphasis makes sense because members of OSS teams may not have the opportunity to form impressions of their teammates through the more traditional types of interactions (i.e. face-to-face work and social interactions) that members of a commercial organization, like Microsoft, would have. As a result, code reviews become even more important for forming impressions of teammates. Conversely, the Microsoft respondents consider the knowledge dissemination aspects more important. Code reviews work as a medium to mentor new teammates about the project design, coding conventions, and available API or libraries.

Similarly, when deciding whether to accept a code review request, the most important factors for OSS respondents are their relationship with the code author and the reputation of the code author. This focus is driven by the desire to maintain current relationships and to improve relationships with reputed developers. Conversely, for Microsoft respondents, when deciding whether to accept a review, the most important factors were the expertise of the code author (i.e., if a developer writes good code that s/he can learn) and the effort required to review the change. When deciding who to invite to review their code, the most important factor was the expertise of the reviewer (i.e., whether s/he has expertise to review that code and will be able

to provide useful feedback).

3.6.6 Effects of Perceived Expertise

A reviewer's perception of the expertise of the code author not only influence the acceptance or rejection of code review requests but also influences the level of scrutiny for the code. First, in terms of accepting incoming code reviews (See section 3.5.3), perceived expertise has mixed influence. Some respondents preferred to review code from experts to learn and to minimize time spent in reviews. Other respondents prioritized reviews from newcomers or focused on areas requiring the most attention. Both of these approaches may be correct, depending upon the expertise of the reviewer.

Second, in terms of the level of scrutiny given to the code, if a reviewer is uncertain about a particular design choice, s/he is more likely to trust the design choice of experts and to question the rationale of non-experts. In addition, if a reviewer does not have adequate time to perform a thorough review of code from an expert author, s/he will approve that code after only a cursory review, based upon an assumption that the expert author implemented correctly as usual. However, an author only receives this status (receiving only cursory reviews) after consistently submitting high-quality code changes.

3.7 Threats to Validity

This section discusses the addressed and unaddressed threats to validity. It is organized around the four common types of validity threats.

3.7.1 Internal validity

Participant selection is the primary threat to internal validity. The subject population consisted of reviewers who had participated in at least 30 code reviews (either as the author or reviewer). It is possible that using a different threshold would have produced different results, but we have no evidence to suggest this situation. In addition, there is the threat that only those subjects who had positive experiences with code review took time to respond to the survey. There is no evidence to suggest that this self-selection occurred. But, even if it did, because the goal of the survey was to gather information about various aspects of code review, those who

had positive experiences could probably provide the best feedback.

3.7.2 Construct validity

The survey design process specifically focused on reducing construct validity threats. This process took approximately eight months and included both expert reviews and multiple pilot tests. The design process included the following bias-reducing practices:

- placing the questions about the topics of interest after the other survey questions to prevent *hypothesis-guessing*,
- presenting the scale questions in random order,
- providing clear definitions of *code review partner* and *non-code review partner* on all relevant pages, and
- carefully wording questions in an unbiased manner.

Third, we conducted multiple reliability and validity tests, with widely-used and highly recommended measures, to ensure construct validity.

3.7.3 External validity

Due to the wide diversity within the OSS community, it is possible that the results may not be representative of all OSS projects. In fact, as most respondents came from well-known, successful OSS projects, they may have been among the higher skilled and more motivated OSS developers. The impacts of code review on software quality and on the social fabric of the team may differ in other types of OSS projects.

In terms of the Microsoft developers, they may not be representative of all commercial organizations. To reduce this potential threat, the respondents came from teams that differ in *development process* (e.g. waterfall vs. agile), *hardware platform* (e.g., mobile, desktop, server, and data center software), *deployment method* (boxed products versus web services), *operating system* (iOS, Windows, Windows Phone, and Linux), *location* (U.S., Europe, and Asia), and *workflow* (e.g., some teams require two reviews on all sign-offs, others are more lax; some want review prior to checkin and test, others do review afterwards; some include testers and development leads on reviews and some do not). In addition, the code review practice at Microsoft is

similar to that used by other large organizations. One frequent misconception is that empirical research within one company or one project is not good enough, provides little value for the academic community, and does not contribute to scientific development. Historical evidence shows otherwise. Flyvbjerg provides several examples of individual cases that contributed to discovery in physics, economics, and social science (Flyvbjerg, 2006). Nonetheless, some biases may remain.

3.7.4 Conclusion validity

The number of responses to each survey was sufficiently large to mitigate any threats arising from small sample sizes. In addition, the Chi-square test (used most frequently in this analysis) does not assume normality in the data. For variables that were not normally distributed (according to the Shapiro-Wilk test), we used non-parametric tests.

3.8 Conclusion

This paper describes the results of two surveys to better understand the practice and motivation for performing code reviews. These results has several implications for researchers and for practitioners. First, although only one-fourth of the code review comments are about functional defects, practitioners should not be discouraged to practice code reviews. Code review offers several other benefits (i.e., knowledge dissemination, relationship building, better designs, and ensuring maintainable code) that are crucial for large scale or long term projects. Interestingly, most code review research focuses on defect detection. These other aspects of code review that are considered more important by the developers have not received much attention. Therefore, these other aspects of code reviews (i.e., relationship building, knowledge sharing, achieving better designs) warrant additional focused research.

Second, the results of these surveys indicate that code reviews have a large impact on relationship building and future collaborations. Carelessness in wording a review comment can lead to negative feelings from the code author and hinder future collaborations. For example, an author is more likely to make the required changes if the reviewer provides constructive criticism and is more likely to argue with the reviewer if the reviewer comments are viewed as an attack. Therefore, reviewers should carefully consider how their review comments will be

heard by the code author. This finding warrants further research in two directions: 1) empirical validation of how the expression of sentiment (i.e., positive or negative) in code review comments influences the code review outcomes and long term collaborations, and 2) how to assist reviewers in articulating appropriate comments during code reviews.

Finally, effective code reviews require a significant amount of effort from the reviewers to thoroughly understand the code. The results of this study suggest that reviewers prefer to review code changes that are simple, self-documenting and easy to comprehend. Authors should keep those code characteristics in mind when submitting code changes for review. This result could also be of interest to program comprehension researchers. The large amount of time devoted to understanding code changes could be improved with appropriate program comprehension techniques.

3.9 References

- Asundi, J. and Jayant, R. (2007). Patch review processes in open source software development communities: A comparative case study. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, 2007. HICSS 2007.*, pages 166c–166c.
- Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721.
- Balachandran, V. (2013). Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 931–940.
- Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2013). The influence of non-technical factors on code review. In *Proceedings of the 20th Working Conference on Reverse Engineering (WCORE)*, pages 122–131.
- Beller, M., Bacchelli, A., Zaidman, A., and Juergens, E. (2014). Modern code reviews in open-source projects: which problems do they fix? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 202–211.

- Bird, C., Carnahan, T., and Greiler, M. (2015). Lessons Learned from Building and Deploying a Code Review Analytics Platform. In *Proceedings of the 12th International Conference on Mining Software Repositories*. IEEE.
- Bosu, A., Carver, J., Guadagno, R., Bassett, B., McCallum, D., and Hochstein, L. (2014). Peer impressions in open source organizations: A survey. *Journal of Systems and Software*, 94(0):4 – 15.
- Bosu, A. and Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. In *Proceedings of the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 133–142.
- Bosu, A. and Carver, J. C. (2014a). How do social interaction networks influence peer impressions formation? a case study. In Corral, L., Sillitti, A., Succi, G., Vlasenko, J., and Wasserman, A., editors, *Open Source Software: Mobile Open Source Technologies*, volume 427 of *IFIP Advances in Information and Communication Technology*, pages 31–40. Springer Berlin Heidelberg.
- Bosu, A. and Carver, J. C. (2014b). Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM ’14, pages 33:1–33:10.
- Bosu, A., Greiler, M., and Bird, C. (2015). Characteristics of useful code reviews: An empirical study. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR ’15, page TBD.
- Bukowski, W. M., Hoza, B., and Boivin, M. (1994). Measuring friendship quality during pre-and early adolescence: The development and psychometric properties of the friendship qualities scale. *Journal of Social and Personal Relationships*, 11(3):471–484.
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum.
- Cohen, J., Brown, E., DuRette, B., and Teleki, S. (2006). *Best Kept Secrets of Peer Code Review*. Smart Bear.

Czerwonka, J., Greiler, M., and Tilford, J. (2015). Code reviews do not find bugs. how the current code review best practice slows us down. IEEE Institute of Electrical and Electronics Engineers.

DeVellis, R. F. (2011). *Scale development: Theory and applications*, volume 26. Sage publications.

Fagan, M. (2002). A history of software inspections. In *Software pioneers*, pages 562–573. Springer.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

Fink, A. (2003). *The survey handbook*, volume 1. Sage Publications.

Flyvbjerg, B. (2006). Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Johnson, P. M. (1998). Reengineering inspection. *Communications of the ACM*, 41(2):49–52.

Johnson-George, C. and Swap, W. C. (1982). Measurement of specific interpersonal trust: Construction and validation of a scale to assess trust in a specific other. *Journal of Personality and Social Psychology*, 43(6):1306.

Lakhani, K., Wolf, B., Bates, J., and DiBona, C. (2002). The boston consulting group hacker survey. *Boston, The Boston Consulting Group*.

Lakhani, K. and Wolf, R. G. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects.

Litwin, M. S. (1995). *How to measure survey reliability and validity*, volume 7. Sage Publications.

- McAllister, D. J. (1995). Affect- and cognition-based trust as foundations for interpersonal cooperation in organizations. *The Academy of Management Journal*, 38(1):pp. 24–59.
- McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 192–201.
- Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49.
- Rempel, J. K., Holmes, J. G., and Zanna, M. P. (1985). Trust in close relationships. *Journal of personality and social psychology*, 49(1):95.
- Rigby, P., Cleary, B., Painchaud, F., Storey, M.-A., and German, D. (2012). Contemporary peer review in action: Lessons from open source development. *IEEE Software*, 29(6):56–61.
- Rigby, P. C. and Bird, C. (2013). Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ES-EC/FSE 2013, pages 202–212.
- Rigby, P. C. and German, D. M. (2006). A preliminary examination of code review processes in open source projects. Technical Report DCS-305-IR, University of Victoria.
- Rigby, P. C., German, D. M., and Storey, M.-A. (2008). Open source software peer review practices: a case study of the apache server. In *Proceedings of the 30th International Conference on Software Engineering*, pages 541–550.
- Rigby, P. C. and Storey, M.-A. (2011). Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 541–550, New York, NY, USA. ACM.
- Robinson, J., Wrightsman, L., and Andrews, F. (1991). *Measures of personality and social psychological attitudes*, volume 1. Academic Press.

Rotter, J. B. (1967). A new scale for the measurement of interpersonal trust1. *Journal of personality*, 35(4):651–665.

Sutherland, A. and Venolia, G. (2009). Can peer code reviews be exploited for later information needs? In *Proceedings of the 31st International Conference on Software Engineering-Companion Volume, 2009. ICSE-Companion 2009.*, pages 259–262.

Votta Jr, L. G. (1993). Does every inspection need a meeting? In *ACM SIGSOFT Software Engineering Notes*, volume 18, pages 107–114.

Wiegers, K. E. (2002). *Peer reviews in software: A practical guide*. Addison-Wesley Boston.

Wohlin, C., Höst, M., and Henningsson, K. (2003). Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer.

APPENDIX

3.A Survey Questions

Q1. (OSS)	Which Free/Open Source project are you most actively involved in?
Q1. (MS)	What product do you currently work on?
Q2.	How many years have you worked in software development?
Q3.	On average, how many people contribute code or review code in a given month for the project?
Q4.	What proportion of the overall code commits in the project undergo peer code review? <input type="radio"/> Less than 10% <input type="radio"/> 11% - 25% <input type="radio"/> 26% - 50% <input type="radio"/> 51% - 75% <input type="radio"/> More than 75%
Q5. (OSS)	Do you receive financial compensation for your participation in the project? <input type="radio"/> Yes <input type="radio"/> No
Q5. (MS)	Do most of the code change reviews that you are asked to participate in come from authors who are at your site or from a distributed site (a site that is in another time zone or more than 100 miles away)? <input type="radio"/> Most come from people that are at a different site than myself <input type="radio"/> Most come from people in the same site as myself <input type="radio"/> Approximately half from people in same site and half from distributed sites
Q6.	Do you think peer code review is important in your project? <input type="radio"/> Yes <input type="radio"/> No
Q7.	Why do you think peer code review is important (or not important) for your project?
Q8.	On the project, how many hours per week, on average, do you spend reviewing other contributors code?
Q9.	From approximately how many different contributors do you review code each week for the project?

Q10.	<p>What proportion of your code commits do you submit for peer code review?</p> <p><input type="radio"/> Less than 10% <input type="radio"/> 11% - 25% <input type="radio"/> 26% - 50% <input type="radio"/> 51% - 75% <input type="radio"/> More than 75%</p>
Q11.	<p>Is the identity of the contributor relevant to you when you decide whether to review a code contribution?</p> <p><input type="radio"/> Yes <input type="radio"/> No</p>
Q12.	<p>Please explain why the identity of the contributor is relevant to you when you decide whether to review a code contribution.</p>
Q13.	<p>When you review poorly written code, does it affect your perception of the code author?</p> <p><input type="radio"/> Yes <input type="radio"/> No</p>
Q14.	<p>Please explain how does poorly written code affect your perception of the code author.</p>
Q15.	<p>When you see poorly written code, how do you help the code reach the level required to be included in the project (if at all)?</p>

Q16.	<p>Please rate the following factors on a 6-point scale based on how strongly each indicates that code you are reviewing is poorly written.</p> <p>(5 - Most important, 4 - Second most important, 3 - Third most important, 2 - Fourth most important, 1 - Fifth most important, 0 - Not in top five)</p> <ul style="list-style-type: none"> -- Poor readability -- Lack of comments -- Does not maintain application integrity -- Poor performance -- Unnecessary complexity -- Lack of modularity (large functions / classes) -- Does not follow coding convention of the project -- Inadequate exception handling -- Duplicated code (Identical or similar code exists in more than one location) -- A large parameter list to a function
Q17.	<p>When you review code of high quality or that has an outstanding approach to solve a problem, does it affect your perception of the code author?</p> <p><input type="radio"/> Yes <input type="radio"/> No</p>
Q18.	<p>Please explain how does high quality or an outstanding approach to solve a problem affect your perception of the code author.</p>
Q19.	<p>Is there anything else you would like to add about this survey or code review process in your project?</p>

Table 3.5: Behavioral Scale Questions

Construct	Item	Question
Trust	trust_1	My communication with him/her is more informal (e.g. unofficial, friendly)
	trust_2	S/he is less likely to intentionally misrepresent my point of view to others
	trust_3	S/he is less likely to take advantage of me (e.g. exploit or deceive for personal benefit)
	trust_4	I am more likely to share my personal information (e.g. feelings, opinions, or achievements) with him/her
Perception of Expertise	expertise_1	It is easier for me to identify whether s/he has the ability to provide help in a specific project area
	expertise_2	I more easily know if s/he is the best person to contribute to a specific project area
	expertise_3	It is easier for me to identify if s/he is the right person to fix a given bug report
	expertise_4	I am more likely to seek his/her help in a project-related area (e.g. coding problem, design decision, task assignment, documentation)
Reliability	reliability_1	I am more comfortable assigning a critical task to him or her
	reliability_2	I am more likely to be satisfied with the results of a task assigned to him or her
	reliability_3	S/he is more likely to complete a project-related task (not just code review) s/he accepts even if it requires a large amount of work
	reliability_4	S/he is more likely to follow project coding and design guidelines
	reliability_5	I am more willing to accept his/her advice
Friendship	friendship_1	I would feel a stronger sense of loss at his/her departure from the project
	friendship_2	S/he has a better understanding of me (e.g. aware of my preferences and feelings)
	friendship_3	S/he is more likely to respond to my mailing list posts
	friendship_4	I communicate more frequently with him/her
	friendship_5	If s/he does something I do not like, I am more likely to talk to him/ her about it.
	friendship_6	I am more likely to consent working together with him/her on a project design task.

CHAPTER 4

IMPACT OF DEVELOPER REPUTATION ON CODE REVIEW OUTCOMES IN OSS PROJECTS

The success of an Open Source Software (OSS) project is affected by the contributions of each individual developer. One of the primary motivations for many OSS developers is the desire to gain an identity and build a good reputation (Lakhani and Wolf, 2005). This motivation drives the amount of effort an OSS developer devotes to the OSS project (Hann et al., 2004) and helps ensure that an OSS developer invest his/her efforts to keep the project on track to be successful (Markus et al., 2000). While a good reputation is an important goal for individual OSS developers, it is not clear what impact a developer's reputation has on project success. If having a good reputation increases the value of an OSS developer's project contributions, then OSS projects that facilitate reputation building should be more successful.

There are different ways to contribute to an OSS project — e.g., posting defects, participating in mailing-list discussions, submitting bug fixes, providing user support, reviewing code, and making code changes. Contributions related directly to code (i.e. making code changes and reviewing code changes) are among the most important types of contributions, because delivering quality software requires contributors to make quality code changes. To ensure the inclusion of only high-quality code into the project codebase, many mature OSS projects have adopted code review as a mandatory quality assurance gateway, thereby elevating the importance of code review. If a developer's reputation can be shown to have a positive impact on the outcome of code review, then we have evidence of the value of developer's reputation on the project outcome as a whole. *Therefore, the objective of this study is to identify how an OSS developer's reputation affects the outcome of his/her code review requests.* Specifically, we explore whether an OSS developer's reputation affects the following aspects of code review: 1) feedback time, 2) review interval, 3) acceptance of code, and 4) number of patches before a code is accepted.

The structure of OSS development communities has often been described as a *core-periphery structure* (Mockus et al., 2002; Ye and Kishida, 2003; Dinh-Trong and Bieman, 2005), with a small number of *core developers* and a larger set of *peripheral developers*. The small set of *core developers* are those who have been involved with the OSS project for a relatively long time and make significant contributions to guide the development and evolution of the project (Ye and Kishida, 2003). The larger set of *peripheral developers* occasionally contribute to the project (Ye and Kishida, 2003), mostly interact with the core developers, and rarely interact with other peripheral developers (Lakhani, 2006). Due to their significant contributions and higher level of interactions, core developers are the most reputed contributors in an OSS community (Ye and Kishida, 2003). Therefore, the core-periphery distinction should be a good proxy for reputation.

In this study, we use social network analysis (SNA) of code review interactions to divide the OSS developers into core and periphery groups. In OSS projects that require all code to undergo peer review, the number of changes a developer has submitted for review and the number of changes s/he has reviewed serve as proxies for his/her level of involvement in the project. Therefore, in these types of projects, analysis of code review interactions can reveal the most influential developers (i.e. core) of an OSS project. Moreover, the results of our prior studies also support using code review social networks to identify the most reputed users. First, the results of a large scale survey of OSS developers showed that code review interactions are highly effective in building positive reputation among the review participants (Bosu and Carver, 2013). Subsequently, we found that due to direct interactions between the developers and lower network centralization, code review social networks have the most favorable characteristics to support building mutual reputation (Bosu and Carver, 2014a).

We create code review social networks of eight popular OSS projects using data mined from Gerrit code review repositories. We develop a novel Core Identification using K-means (CIK) clustering approach to divide the OSS developers into core and periphery groups. Working on the assumption that core developers have a better reputation than peripheral developers, we compared the code review outcomes of core developers to the code review outcomes of the peripheral developers to investigate the effects of reputation. We have published initial results

of this work (Bosu and Carver, 2014b). This paper extends the prior work by providing more in-depth analyses of that work and adding three research hypotheses. The main contributions of this work are:

- empirical evidence regarding the effect of OSS developers' reputation on code review outcome,
- an overview of OSS social networks based on code review interactions, and
- a novel approach to group OSS developers into core and periphery.

The rest of the chapter is organized as follows. Section 4.1 provides background about code review and social network analysis. Section 4.2 introduces the research questions. Section 4.3 describes our novel approach to identify the core nodes. Section 4.4 describes the analysis and results. Section 4.5 discusses the implications of the results. Section 4.6 addresses the threats to validity. Finally, Section 4.7 concludes the chapter.

4.1 Background

This section provides background information on three topics relevant to this study: code review, social network analysis, and core-periphery structure in OSS projects.

4.1.1 Contemporary Code Review

Code review (a.k.a. peer code review), the process of analyzing code written by a teammate (i.e. a peer) to judge whether it is of sufficient quality to be integrated into the main project codebase, has recently been adopted by many mature successful OSS projects. According to Bacchelli and Bird (2013), contemporary code reviews are more informal and more common than the Fagan-style inspection (Fagan, 1976). In OSS and commercial organizations, code reviews are also increasingly supported by tools (Rigby and Bird, 2013).

One of the prominent code review tools is Gerrit¹. Gerrit captures the following detailed information about the code review. Developers submit a *patchset* (i.e. all files added or modified in a single revision) to a Gerrit repository for review. The Gerrit interface displays the changes

¹ <https://code.google.com/p/gerrit/>

side-by-side, allows the reviewers to insert inline comments, and allows the author to respond to those comments . Gerrit documents all of these interactions. If reviewers requests changes, the author can make those changes and upload a new patchset. This process repeats until the reviewers approve the patchset, which can then be merged into the main project branch.

4.1.2 Social Network Analysis

A social network is a theoretical construct that represents the relationships between individuals. These networks are typically modeled as a graph in which nodes represent individuals and edges represent the relationship between individuals. SNA identifies social relationships and interaction patterns among individuals in a community based on the idea that these patterns represent an important aspect of the lives of those individuals (Freeman, 2004). These social interaction patterns also affect some important features of a community (e.g., efficiency when performing a task, group dynamics, and leadership) (Wasserman, 1994). Therefore, SNA is common approach for identifying and studying relationships in a variety of domains, e.g., sociology, biology, anthropology, communication studies, information science, organizational studies, and psychology.

Software engineering researchers have used SNA techniques to understand different types of developer interactions in OSS projects. First, using *code commit* interactions, studies show that developers who work on the same file are more likely to interact in the mailing-list (Bird et al., 2008), and SNA techniques can identify the influence of a sponsoring company on the development process, release management, and leadership turnover (Martinez-Romo et al., 2008). Second, using *bug fixing* interactions, studies found decentralized communication patterns in larger projects (Crowston and Howison, 2005) and a stable core-periphery structure with decreasing group centralization over time (Long and Siau, 2007). Third, using *mailing-list* interactions, studies found developers having higher status than non-developers (Bird et al., 2006) and the core developers having disproportionately large share of communication with the peripheral developers (Oezbek et al., 2010). The application of SNA techniques in our study differs from those prior studies in two key ways: 1) using source data from a different types of interaction (i.e. code review), and 2) using SNA techniques for measuring developer reputation.

4.1.3 Core-Periphery Structure in OSS Communities

A social network that exhibits the **Core-Periphery structure** has a set of densely connected *core* nodes and a set of more loosely connected *periphery* nodes (Borgatti and Everett, 2000). For example, Figure 4.1 shows a toy network of 10 nodes with a core-periphery structure. The four central nodes (i.e. 1, 2, 3, and 4) have connections with each other and forms the dense core. The remaining six nodes are loosely connected with the core nodes and do not have edges between them.

Several studies has focused on characterizing the core-periphery structures of OSS development communities. Fielding was the first to describe the core developer group and their roles in the Apache project (Fielding, 1999). Later studies found that while the core contains only 3-25% of the developers, these developers contribute 40-90% of the code (Mockus et al., 2002; Robles et al., 2009; Lee and Cole, 2003; Lakhani, 2006; Dinh-Trong and Bieman, 2005). Although, the number of core developers in a project may not vary much, the members belonging the core change over time (Robles et al., 2009). However, frequent core membership turnover may not be good for the sustainability of an OSS project. Long-term core members may have detailed knowledge about the design of the project and are often able to perform maintenance tasks to reduce structural complexity of the project (Terceiro et al., 2010). While most of the prior studies focused on estimating the number of core developers, their contributions, and their roles in the project, we focus on developing a method to identify the core developers and how their code review outcomes differ from those of the peripheral developers.

4.2 Research Hypotheses

The high-level hypothesis of this study is, “*A Developer’s reputation affects the outcome of his/her code review requests*”. Specifically, we use the core-periphery distinction as our measure of reputation. That is, developers in the core have a better reputation than those in the periphery. Therefore, this hypothesis really suggests that aspects of the code review process are affected by whether the code author is a member of the *core* or a member of the *periphery*. The following subsections decompose this high-level hypothesis into four detailed hypotheses.

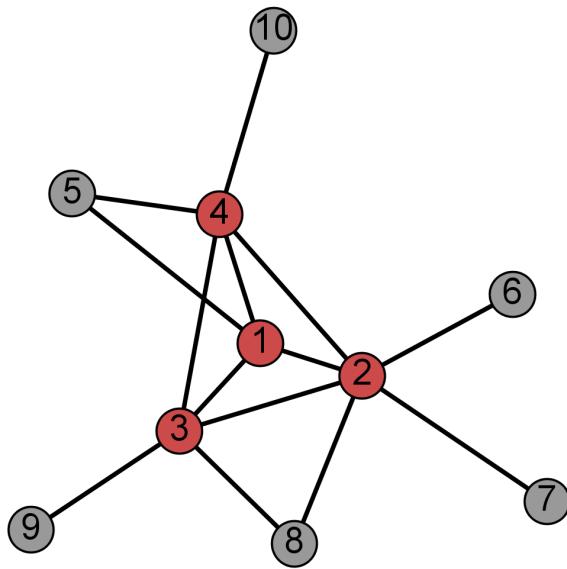


Figure 4.1: A small network with core-periphery structure (adapted from Borgatti and Everett (2000)). Core nodes are colored using red.

4.2.1 First Feedback Interval

We define *First Feedback Interval* as the amount of time from the submission of a code review request in Gerrit until the first real review comment (i.e. not an automated comment). There are several factors that suggest that core developers should have a shorter first feedback interval than the peripheral developers:

- They should know which teammates are most appropriate for reviewing a particular change and can add them as potential reviewers;
- Their position in the core group may lead reviewers to provide a less thorough, less time-consuming review; and
- Their prior interactions and good relationships may encourage other members to prioritize their code for review.

Therefore, we pose the following hypothesis:

H1 *The First Feedback Interval is shorter for review requests from core developers than for requests from peripheral developers.*

4.2.2 Review Interval

Review Interval is the time from the beginning to the end of the review process (Rigby et al., 2008). We define the review process to be complete when the patchset is ‘Merged’ to the main project branch or is ‘Abandoned’. For similar reasons as listed in Section 4.2.1, core developer may have a shorter overall review interval. Moreover, due to their familiarity with the codebase and their prior interactions with the reviewers, core developers may be able to understand and make the suggested changes more quickly than the peripheral developers. Therefore, we pose the following hypothesis:

H2 *The Review Interval is shorter for review requests from core developers than for requests from peripheral developers.*

4.2.3 Code Acceptance Rate

A developer’s *Code Acceptance Rate* is the ratio between the number of review requests submitted and the number ‘Merged’. Due to more familiarity about the project design, we suggest that core developers are able to write acceptable code more frequently than the peripheral developers. Moreover, their positions in the core help them influence the project development direction (Gacek and Arief, 2004), to integrate their suggested features into the project more often. Therefore, we pose the following hypothesis:

H3 *Core developers have higher code acceptance rate than the peripheral developers.*

4.2.4 Number of Patchsets per Review Request

If a reviewer identifies a problem during code review, the author must upload a new patchset to fix that problem. The reviewer reviews the new patchset and either accepts it or requests further modifications. This process repeats until the reviewer is satisfied with the changes and agrees the code is ready to be merged. Because core developers have already made many code changes, they should be familiar with the project requirements, design, and coding guidelines. Therefore, they should be able to write code that can be merged in fewer tries (i.e. fewer patchsets). In addition, because they are in the core, other developers may approve their patchsets more quickly. Therefore, we pose the following hypothesis:

H4 *Core developers are able to get code changes accepted through lower number of patchsets than the peripheral developers.*

4.2.5 Number of Review Comments

Reviewers provide comments when they identify issues or have suggestions to improve the code. Because core developers are usually more experienced than the peripheral developers, their code changes should have fewer issues. Therefore, we hypothesize:

H5: *Core developers write code with less number of issues (i.e. less review comments) than the peripheral developers.*

4.2.6 Patchset size

Core developers have made significant contributions to their projects and tend to work on the most critical code changes. Peripheral developers tend to work on minor or trivial changes as they prove themselves and become familiar with the project. Therefore, we pose the following hypothesis:

H6: *Core developers submit larger patchsets than peripheral developers.*

4.2.7 Number of Reviewers per Review Request

Because core developers have built higher level of trust and reputation through their contributions over time, their code changes may be subject to less scrutiny and require approval from fewer reviewers. Therefore, we hypothesize:

H7: *Core developers' changes require verification from fewer reviewers than changes from peripheral developers'.*

4.3 Core Identification using K-means (CIK)

The following subsections describe the collection and preparation of data, construction of the social networks, detection of core and peripheral developers, and evaluation of the approach.

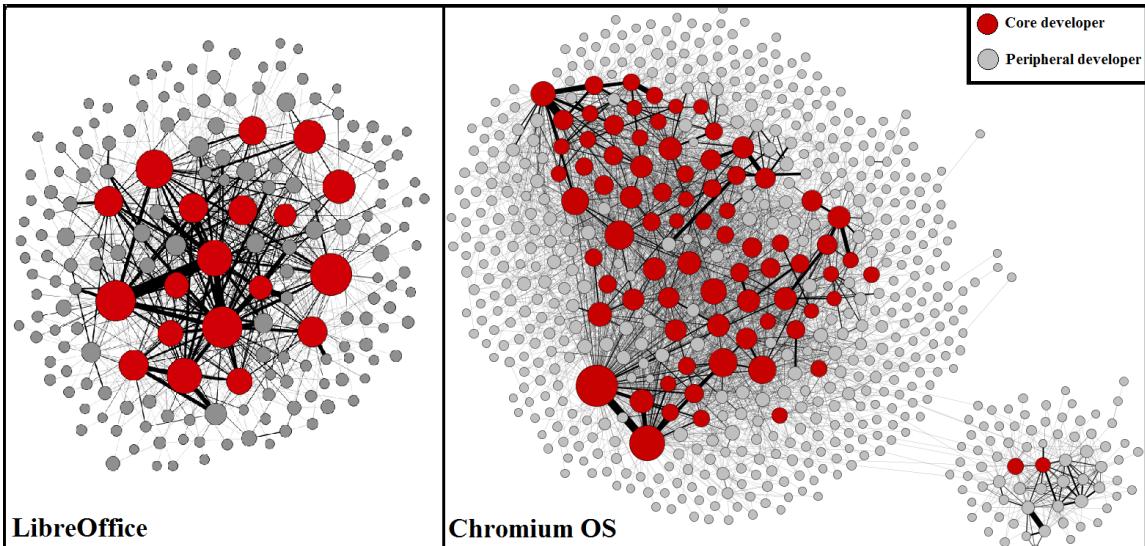


Figure 4.2: Code Review Social Network Diagram of two OSS Projects

4.3.1 Data Collection and Preparation

Following a similar approach as Mukadam et al. (2013), we developed the *Gerrit-Miner* tool that can mine the publicly accessible code review data in a Gerrit repository. In January of 2014, we used *Gerrit-Miner* to mine the code review repositories of eight popular OSS projects (Table 4.2) with regular code review activity. We mined all completed code review requests (i.e. those marked as either ‘Merged’ or ‘Abandoned’). A manual inspection of the comments posted by some accounts (e.g., Qt Sanity Bot or BuildBot) suggested that those accounts were automated bots rather than humans. These accounts typically contain one of the following keywords: ‘bot’, ‘auto’, ‘CI’, ‘Jenkins’, ‘integration’, ‘build’, or ‘verifier’. Because we wanted only code review comments from actual reviewers, we excluded these bot accounts after a manual inspection confirmed that the comments were automatically generated.

Projects included in this study require that each code change pass through a mandatory code review before inclusion in the project codebase. Therefore, the number of ‘Merged’ review requests is a measure of the number of commits by an OSS developer that were integrated into the project codebase. The sum of the number of ‘Merged’ and ‘Abandoned’ requests is a measure of the total number of patchsets submitted by an OSS developer. These two measures allow us to use code review data to calculate the code contributions of a developer.

4.3.2 Building Social Networks

Based on the mined data, we calculated the number of code review interactions between each pair of developers in each project. Using this data, we generated social network graphs as undirected, weighted graphs where nodes represent developers and edge weights represent the quantity of interactions between those developers. For example, if Developer A has commented on 10 code review requests posted by Developer B and Developer B has commented on 15 code review requests posted by Developer A, the weight of the edge between their nodes would be 25. Conversely, if Developers A and B had not commented on each others' code review requests, then no edge would exist.

Based on this data, Figure 4.2 shows the social network structure for two of the eight projects (LibreOffice and Chromium OS) as generated by Gephi (Bastian et al., 2009). Node size visualizes node degree (i.e. the number of attached edges). Node color indicates whether the node is in the core or periphery set (see Section 4.3.3). Edge widths are based on edge weights. To make the diagrams more readable, we removed low-weight edges and then excluded isolated nodes.

4.3.3 Core-Periphery Detection using the CIK Approach

While Borgatti and Everett's statistical approach based on the density of links between nodes is the most common approach to distinguish core nodes from periphery nodes (Borgatti and Everett, 2000), it is not applicable in this study. Borgatti and Everett's approach works well for social network with single core group (e.g. LibreOffice - left side of Figure 4.2). But, it does not work well for networks with multiple core groups (e.g. Chromium OS - right side of Figure 4.2). In fact, the authors suggest that networks with multiple cores be split into multiple networks and analyzed separately (Everett and Borgatti, 2000). In practice, it may be difficult to divide the social networks of the OSS projects in our sample into separate networks with clearly distinguishable core and periphery. For example, the Chromium OS project has two clear groups (i.e. the big cluster on left and small cluster on right). However, there are several groups within the big cluster that are difficult to separate. In this graph, Borgatti and Everett's approach is not useful.

Therefore, we developed a new approach for partitioning the OSS social networks into

core and periphery based on *centrality measures* (i.e. measures of the relative importance of a node within a graph). Because core developers are typically the most important developers in an OSS project, we can use centrality measures to find the nodes that represent those core developers. There are a number of widely used centrality measures: degree, betweenness, closeness, eigenvector, PageRank, and eccentricity. Each measure calculates centrality in a different way and has a different interpretation of a central node. To include the best of all options and reduce the bias from any one centrality measure, we used the K-means clustering algorithm² (MacQueen et al., 1967) to combine all six measures into one new measure. We call this approach *Core Identification using K-means (CIK)*. In a prior article, we have detailed the evaluation of the CIK approach (Bosu and Carver, 2014b).

Table 4.1 provides an overview of the six centrality measures and their interpretation within an OSS social network. We used Gephi to calculate the six centrality measures for each node in each graph. Then, we used the SPSS implementation of the k-means clustering algorithm to partition the nodes into core and periphery groups based on the six centrality scores. Table 4.2 provides a description of the core-periphery partitions for the eight projects in this study.

4.4 Data Analysis and Results

The following subsections describe the process we used to analyze the differences between the core and peripheral developers (Section 4.4.1) and then the results for the seven hypotheses.

4.4.1 Analysis Approach

In our prior study, we observed that most review requests receive their first feedback within 2-4 hours, with some outliers taking up to a week. Similarly, most code review requests had a review interval of couple of days, with a few outliers taking much longer (Bosu and Carver, 2012). In the current study, we observed similar trends. Again, we also observed highly skewed distributions for code churn. To avoid bias in the skewed distributions for first feedback interval, review interval and code churn (i.e. long tails), we used median for the central tendency.

² K-means clustering partitions n observations into k mutually exclusive clusters in which each observation belongs to the cluster with the nearest mean. K-means treats each observation as an object having a location in space. It finds a partition in which objects within each cluster are as close to each other as possible and as far from objects in other clusters as possible.

Table 4.1: Social Network Centrality Measures

Metric	Definition	Interpretation
Degree Centrality	<i>Degree Centrality</i> is the number of edges incident upon a vertex (Freeman, 1977). The degree centrality of a vertex v , for a given graph G , is defined as: $C_D(v) = \deg(v)$.	Degree centrality is an indication of how many other developers each developer interacts with. Core developers usually interact with a large number of other developers resulting in a higher degree centrality.
Betweenness Centrality	The <i>Betweenness Centrality</i> of a vertex is a measurement of the number of shortest paths traversing that vertex (Freeman, 1977). The betweenness centrality of vertex v in graph G is defined as: $C_B(v) = \sum_{s \neq v \neq t} \frac{\delta_{st}(v)}{\delta_{st}}$, where $\delta_{st}(v)$ is the number of shortest paths from s to t going through v , and δ_{st} is the total number of shortest paths between s and t .	Betweenness centrality can be interpreted as a measurement of the importance of a developer within a network, because developers with a high betweenness centrality are intermediate nodes for the communication of the other developers.
Closeness Centrality	The <i>Closeness Centrality</i> of a vertex is a measurement of its proximity to the rest of the vertices in the network (Sabidussi, 1966). The higher the value, the closer that vertex is to the others (on average). The closeness centrality of vertex v in graph G is defined as: $C_C(v) = \frac{1}{\sum_{t \in G} d_G(v, t)}$, where $d_G(v, t)$ is the minimum distance between v and t .	Closeness centrality gives an indication of how quickly a developer can reach the entire community.
Eigenvector Centrality	<i>Eigenvector Centrality</i> is a measure of the importance of a vertex in a network. It assigns relative scores to all vertices in the network based on the principle that connections to high-scoring vertices contribute more to the score of the vertex in question than equal connections to low-scoring vertices. Eigenvector centrality is defined as the principal eigenvector of the adjacency matrix defining the network. The defining equation of “an eigenvector is $\lambda v = Av$ where A is the adjacency matrix of the graph, λ is a constant (the eigenvalue) and V is the eigenvector. The equation lends itself to the interpretation that a node that has a high eigenvector score is one that is adjacent to nodes that are themselves high scorers” (Bonacich, 2007).	Eigenvector centrality gives an indication of a developer’s influence over the network.
PageRank Centrality	<i>PageRank Centrality</i> is based on Google’s PageRank algorithm. PageRank is a variant of Eigenvector Centrality and is based on the same basic concept. The key difference in PageRank is the use of a random surfer model based on a random decay factor (d -called damping factor, usually 0.85), which means that a random surfer would stop following links on the existing node with probability d and teleport to a new node. (Page et al., 1999).	PageRank also gives an indication of a developer’s influence over the network.
Eccentricity	<i>Eccentricity</i> is a measure of how far a vertex is from the most distant vertex. A high eccentricity means that the most distant vertex in the network is a long way away, and a low eccentricity means that the most distant vertex is actually quite close (Newman, 2010). The eccentricity of a node v in graph G is defined as: $C_E(v) = \text{Max}_{t \in G} d_G(v, t)$, where $d_G(v, t)$ is the minimum distance between v and t .	A central developer should have lower eccentricity as they will have direct communication ties with many other developers.

Table 4.2: Core-Periphery partitioning of the projects

Project	Domain	Technol- ogy	Using Gerrit since	Re- quests mined*	Total # of devel- opers	Size of the Core	Size of the periph- ery	% of core mem- bers	% of code com- mits by the core	% of the code reviews by the core
Chromium OS	Operating System	C, C++	February, 2011	49,853	642	79	533	12.9%	64.7%	72.5%
ITK / VTK	Visualization Toolkit	C++	August, 2010	13,207	244	19	225	7.8%	57.0%	77.2%
LibreOffice	Office Application Suite	C++	March, 2012	6,347	207	20	187	9.7%	37.6%	88.0%
OmapZoom	Mobile Development Platform	C	February, 2009	32,930	642	34	608	5.1%	34.3%	60.2%
OpenStack	Cloud Computing Software	Python, JavaScript	July, 2011	59,125	1,880	128	1,752	6.9%	53.6%	66.0%
OVirt	Virtual Machine management	Java	October, 2011	21,316	193	20	173	10.4%	51.3%	61.1%
Qt Project	UI framework	C, C++	May, 2011	71,732	888	63	825	7.1%	55.9%	66.1%
Typo3	Content Management System	PHP, JavaScript	August, 2010	24,374	387	30	357	7.8%	56.3%	71.0%

*Mined during January, 2014

For each developer we calculated the median first feedback interval, median review interval, acceptance rate, average number of patchsets, median code churn, average number of reviewers, and average number of comments per review request. Because, the Shapiro-Wilk's normality test indicated that the distributions of all metrics significantly differed from a normal distribution, we used non-parametric hypothesis tests (Mann-Whitney U) for each of the hypotheses introduced in Section 4.2, and report median for the groups.

In this paper, we use Beanplots (Kampstra et al., 2008) to visualize and compare the distribution density for multiple samples along the y-axis. Beanplots are best for a large range of non-normal data as they show the entire distribution (they essentially show the full distribution drawn vertically, and show whether there are peaks and valleys in a distribution). We used Beanplots to visualize the highly skewed distributions of first feedback interval, review interval, and patchset size. The horizontal line in the middle indicates median of the distribution. We use boxplots to visualize highly concentrated data (i.e., acceptance rate, number of comments, number of reviewers and number of patchsets).

Table 4.3: Statistical Results for H1: First interval

Project	First interval (hours)		Ratio	
	Core	Periphery		
Chromium OS	1.02	2.78	2.73	p<.001*
ITK/VTK	3.47	18.61	5.36	p<.001*
LibreOffice	6.36	17.63	2.77	p<.001*
OmapZoom	0.64	1.72	2.69	p=.01*
OpenStack	2.47	4.44	1.8	p<.001*
OVirt	5.56	11.01	1.98	p=.001*
Qt Project	1.72	5.23	3.04	p<.001*
Typo3	0.04	0.16	4	p=.003*

* Indicates a statistically significant difference

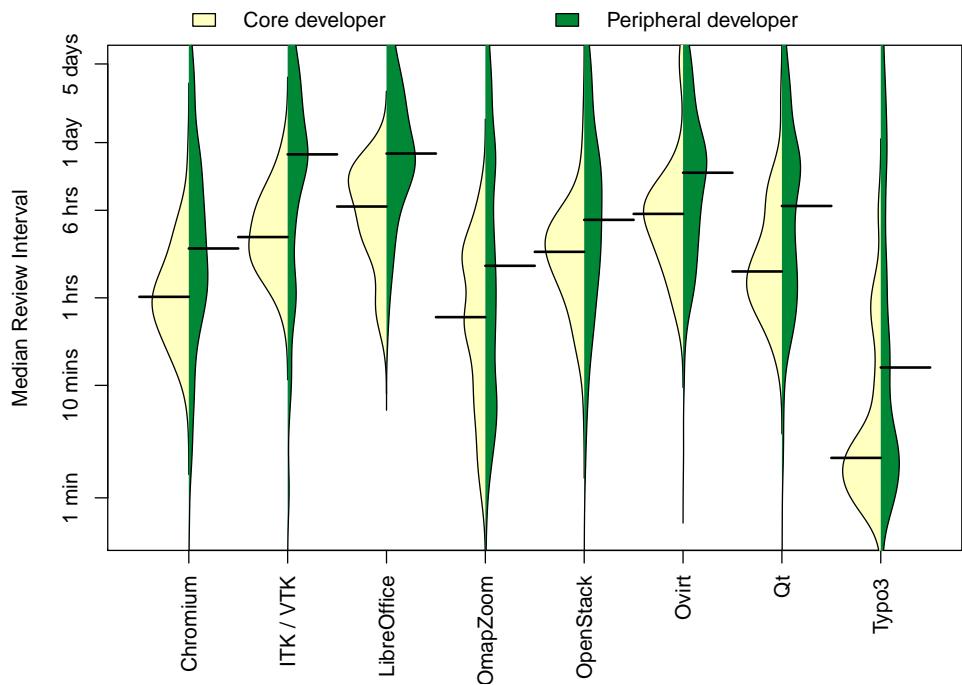


Figure 4.3: First Feedback Intervals

4.4.2 H1: First Feedback Interval

Table 4.3 show the median first feedback interval for the *core* and *peripheral* developers of each project. The first feedback interval is significantly lower for the *core* developers, supporting H1. The beanplots (Figure 4.3) show flatter distributions (i.e., lower peaks), which indicates more variations but very long tails for the peripheral developer, which indicates that many peripheral developers had to wait very long for first feedback. Additionally, the ‘Ratio’ column in Table 4.3 shows that the *peripheral* developers waited 1.8 to 6 times (or 1 to 12 hours) longer for the first feedback on their code review requests.

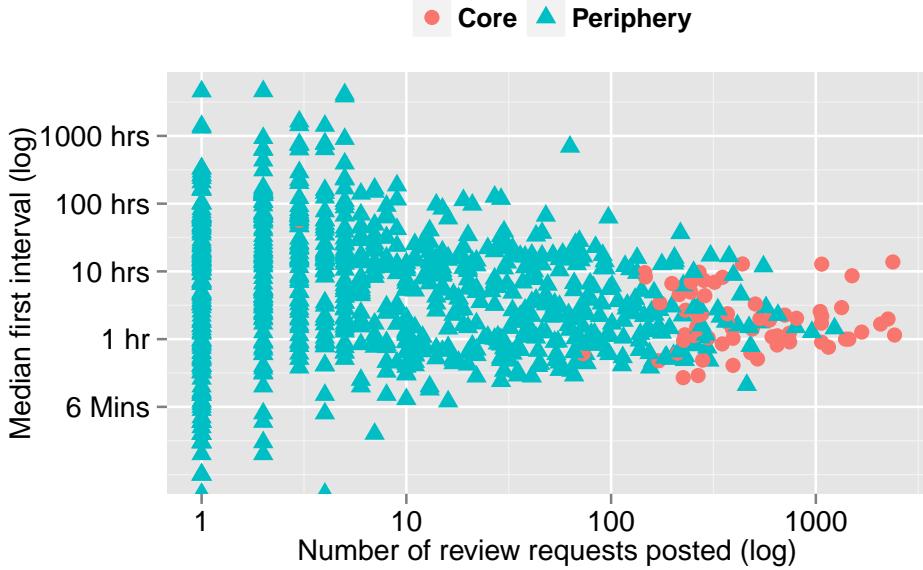


Figure 4.4: Experience vs. First feedback interval (Qt project)

To provide more insight into this result, we analyzed whether a developer's project experience affected the first feedback interval. Figure 4.4 shows a representative example of a scatter-plot (for the Qt project) comparing number of posted review requests (a proxy for project experience) and first feedback interval. Consistent with the previous result, the core developers have posted more review requests and have a lower median first interval. In general, most of the developers posting less than 10 review requests have higher first feedback intervals. Often those who have posted less than 10 review requests are prospective newcomers who have yet to gain recognition within the community. Most of the peripheral developers are prospective newcomers (ranging from 41% in OVirt to 85% in LibreOffice). While a few prospective newcomers have similar first feedback intervals as the core developers (less than 10 hours), the majority had much longer first feedback intervals (100 to 1000 hours). Therefore, prospective newcomers with little or no recognition may have to wait longer for the first feedback on their posted review requests than someone who has more experience.

4.4.3 H2: Review Interval

Table 4.4 shows the median review intervals for the *core* and *peripheral* developers on each project. The median review interval is significantly lower for the *core* developers on all projects, supporting H2. The beanplots (Figure 4.5) have lower peaks but very long tails for the peripheral developer, indicating long review intervals for many peripheral developers.

Table 4.4: Statistical Results for H2: Review interval

Project	Review interval (hours)		Ratio	
	Core	Periphery		
Chromium OS	13.59	27.47	2.02	p<.001*
ITK/VTK	22.62	95.40	4.22	p<.001*
LibreOffice	11.37	39.25	3.45	p<.001*
OmapZoom	73.59	211.62	2.7	p<.001*
OpenStack	56.52	142.21	2.52	p<.001*
OVirt	41.38	148.71	3.59	p<.001*
Qt Project	25.32	72.36	2.86	p<.001*
Typo3	3.99	75.83	19	p<.001*

* Indicates a statistically significant difference

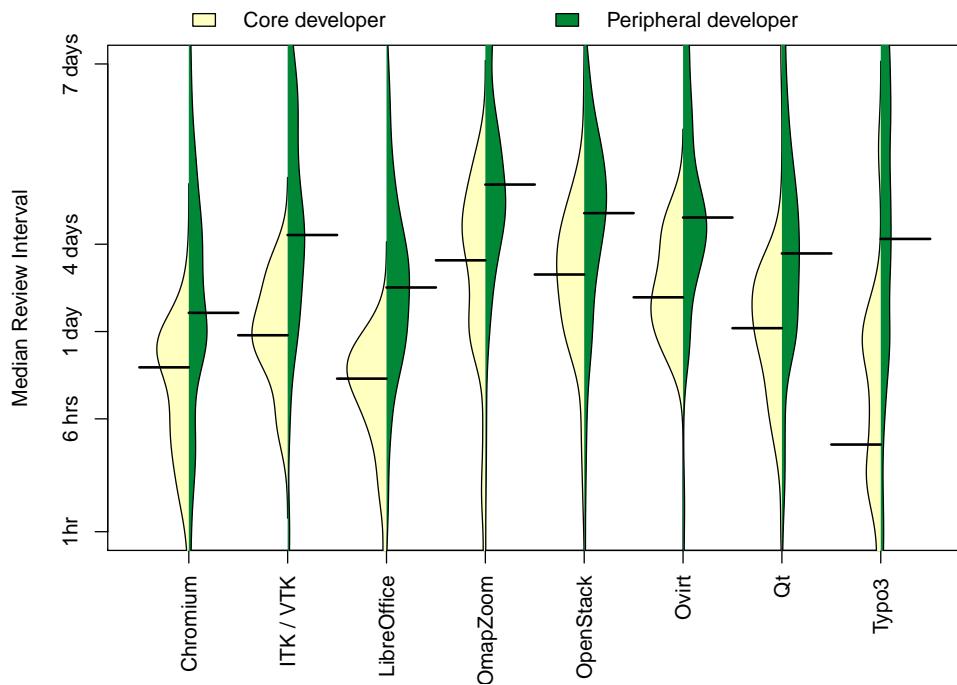


Figure 4.5: Review Intervals

Additionally, the ‘Ratio’ column in Table 4.4 shows that the *peripheral* developers waited 2 to 19 times (or 12 to 96 hours) longer than the *core* developers for the review process to complete.

Figure 4.6 shows a representative example scatter-plot (for the Chromium project) comparing experience (measured by number of review requests posted) to median Review Interval. Similar to H1, most of the peripheral developers that have posted less than 10 requests have longer review intervals. Therefore, peripheral developers not only have to wait longer for the first feedback but also they have to wait longer to complete the review process.

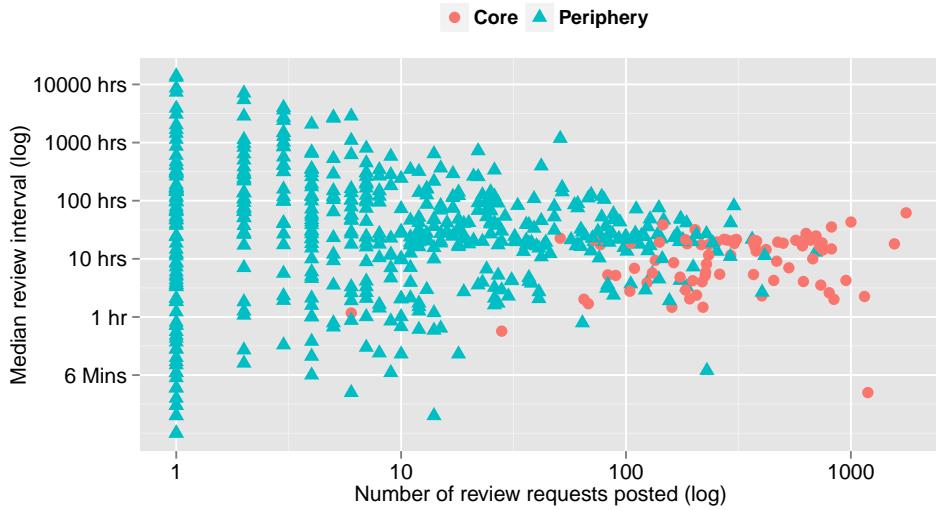


Figure 4.6: Experience vs. Review interval (Chromium OS)

Table 4.5: Statistical Results for H3: Acceptance rate

Project	Acceptance rate		Ratio
	Core	Periphery	
Chromium OS	87.9%	55.7%	p<.001*
ITK/VTK	81.7%	25.0%	p<.001*
LibreOffice	93.3%	66.7%	p<.001*
OmapZoom	69.6%	50.0%	p<.001*
OpenStack	85.2%	42.8%	p<.001*
OVirt	91.2%	70.4%	p<.001*
Qt Project	85.5%	50.0%	p<.001*
Typo3	92.3%	50.0%	p<.001*

* Indicates a statistically significant difference

4.4.4 H3: Acceptance Rate

Table 4.5 and Figure 4.7 show the average acceptance rate for the *core* and *peripheral* developers on each project. The acceptance rate is significantly higher for the *core* developers on all projects, supporting H3.

Figure 4.8 shows an example scatter-plot (for OVirt) comparing experience (measured by number of review requests) and acceptance rate. The developers who have posted less than 10 review requests have the lowest acceptance rates. Therefore, peripheral developers, especially those who have posted less than 10 prior review requests have a lower probability of having their changes accepted.

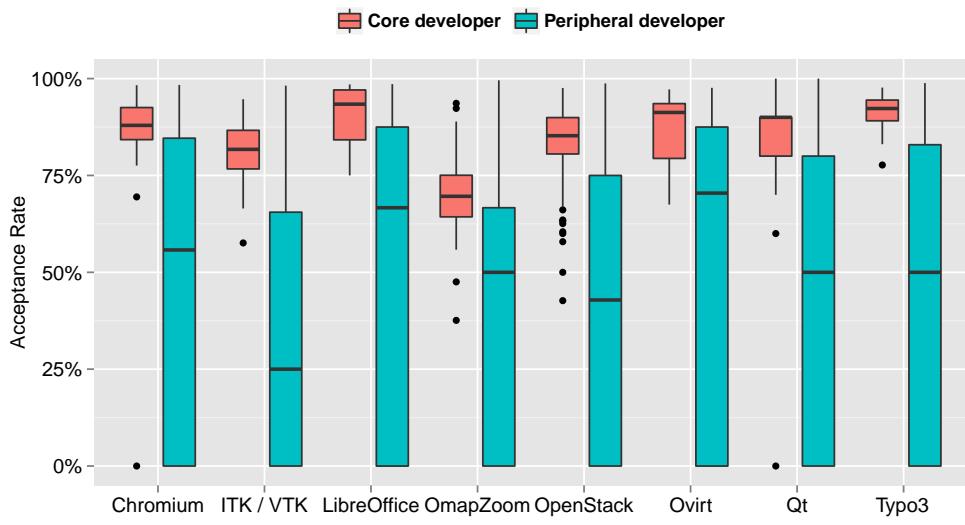


Figure 4.7: Acceptance rate

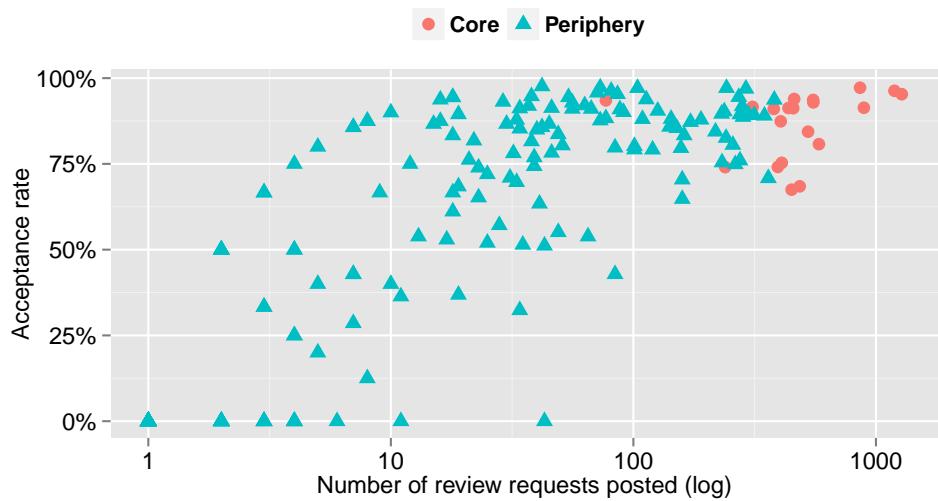


Figure 4.8: Experience vs. Acceptance Rate (OVirt)

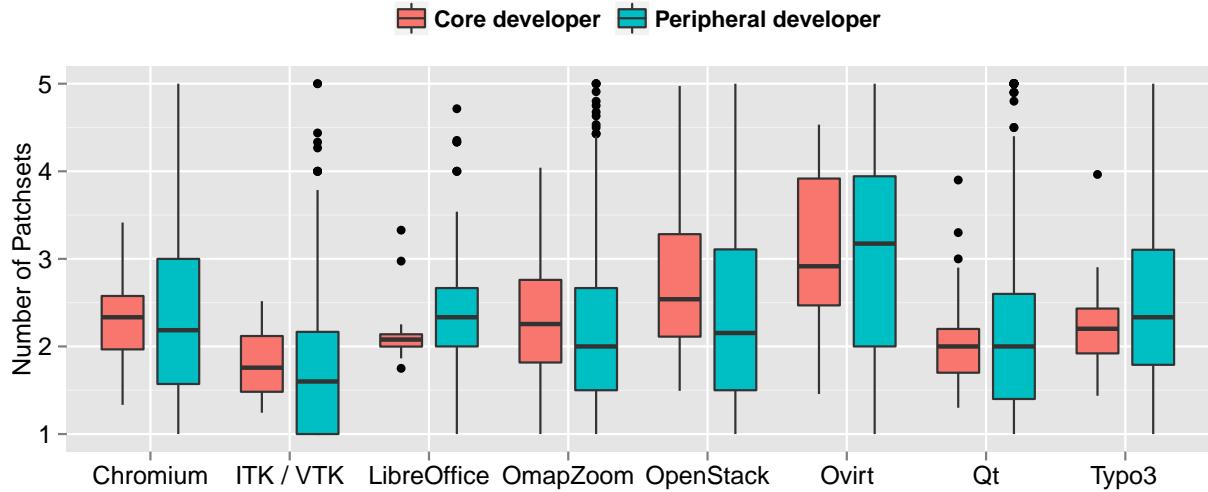


Figure 4.9: Average number of patchsets per review request

Table 4.6: Statistical Results for H4: Number of patchsets

Project	Number of patchsets		p=
	Core	Periphery	
Chromium OS	2.33	2.28	.747
ITK/VTK	1.76	1.67	.265
LibreOffice	2.08	2.33	.02*
OmapZoom	2.26	2.00	.301
OpenStack	2.58	2.42	.016*
OVirt	3.57	3.55	.872
Qt Project	1.98	2.00	.889
Typo3	2.20	2.35	.143

* Indicates a statistically significant difference

4.4.5 H4: Number of Patches per Review Request

Table 4.6 and Figure 4.9 show the average number of patchsets required for a change to be accepted for the *core* and *peripheral* developers on each project. For each project, the *core* developers required fewer patchsets than the *peripheral* developers, although the result is significant for only two projects. This result is inconclusive overall for H4.

4.4.6 H5: Number of Review Comments

Table 4.7 and Figure 4.10 show the average number of comments received for a review request for the *core* and *peripheral* developers on each project. The difference is statistically significant for only half of the projects. For the other four, the results are mixed, but not significant in either direction. Therefore, the result for H5 is inconclusive.

Table 4.7: Statistical Results for H5: Number of comments

Project	Number of review comments		
	Core	Periphery	
Chromium OS	2.27	3.00	p=.000*
ITK/VTK	2.47	2.75	p=.411
LibreOffice	2.40	2.66	p=.002*
OmapZoom	5.27	5.16	p=.417
OpenStack	5.25	5.00	p=.115
OVirt	3.59	5.87	p=.000*
Qt Project	3.39	4.00	p=.002*
Typo3	4.93	5.25	p=.393

* Indicates a statistically significant difference

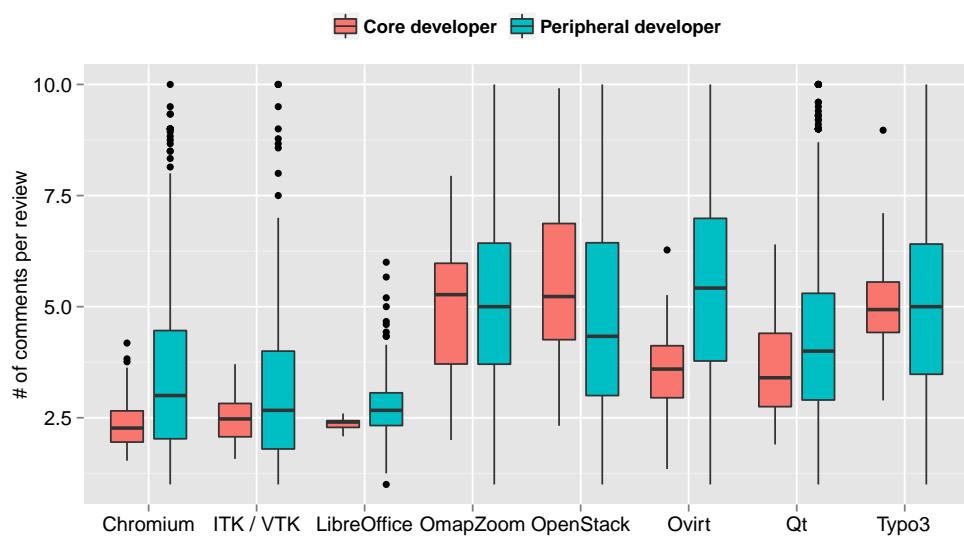


Figure 4.10: Number of review comments

Table 4.8: Statistical Results for H6: Number of patchset size

Project	Patchset size		
	Core	Periphery	
Chromium OS	18	30	p<.001*
ITK/VTK	22	50	p=.047*
LibreOffice	21	53	p<.001*
OmapZoom	15	24	p<.001*
OpenStack	27	37	p<.001*
OVirt	21	23	p=.311
Qt Project	14	27	p<.001*
Typo3	14	17	p=.278

* Indicates a statistically significant difference

To understand this result better, we compared different attributes of the projects such as: number of developers, % of core members, % of contributions by core, and project sponsorship³. We chose these attributes because they can possibly indicate the average review load for a developer, which may influence thoroughness of reviews. For example, if the percentage of core members is low then core developers will have a higher review load because they perform the majority of the reviews. However, we did not observe any trend in a particular set of projects (i.e., projects that supported this hypothesis or projects that did not) that can possibly provide us directions for further analysis. A more in-depth qualitative analysis of code reviews from those projects may provide possible explanations.

4.4.7 H6: Patchset size

In this study, we determine patchset size using the number of lines added, modified or deleted for a code change. Table 4.8 show the median patchset sizes for the *core* and *peripheral* developers of each project. For all projects, the median patchset size is lower for *core* developers, refuting H6. This difference is statistically significant for six projects. The beanplots (Figure 4.11) show flatter distributions (lower peaks) as well as very long tails for the peripheral developer, which indicates that many peripheral developers work on larger patchsets.

Figure 4.12 shows an example scatter-plot (for OmapZoom) comparing experience (measured by number of review requests) and median patchset size. Many of the prospective newcomers (experience less than 10) submitted patchsets which were larger than 100 lines. This

³ whether the project is sponsored by a company, e.g., Chromium OS is sponsored by Google, and OVirt is sponsored by RedHat

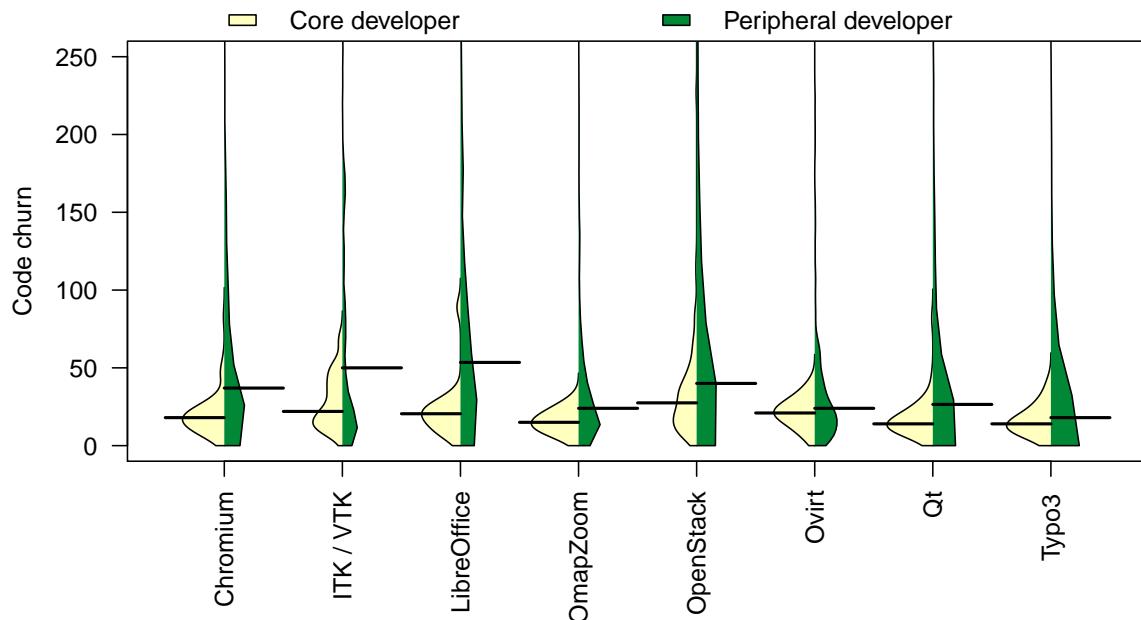


Figure 4.11: Patchset size

result is surprising because it contradicts prior research that suggests a newcomer is more likely to become part of a project community if he or she starts by submitting small patchsets to fix defects (Von Krogh et al., 2003; Herranz et al., 2006; Ducheneaut, 2005). Unsurprisingly, the developers who have posted less than 10 review requests have the lowest acceptance rates. These results also partially explain high number of prospective newcomers that failed to continue their contribution to the OSS projects (Ducheneaut, 2005).

4.4.8 H7: Number of Reviewers per Review Request

Table 4.9 and Figure 4.13 show the average number of reviewers that review code changes from the *core* and *peripheral* developers on each project. The results show statistically significant difference for only one project (Ovirt). Therefore, the result for H7 is inconclusive.

4.5 In-depth Analysis

To gain further insights of the results, following subsections analyzes inter/intra group collaborations, longitudinal analysis of developers reputation formation in a community.

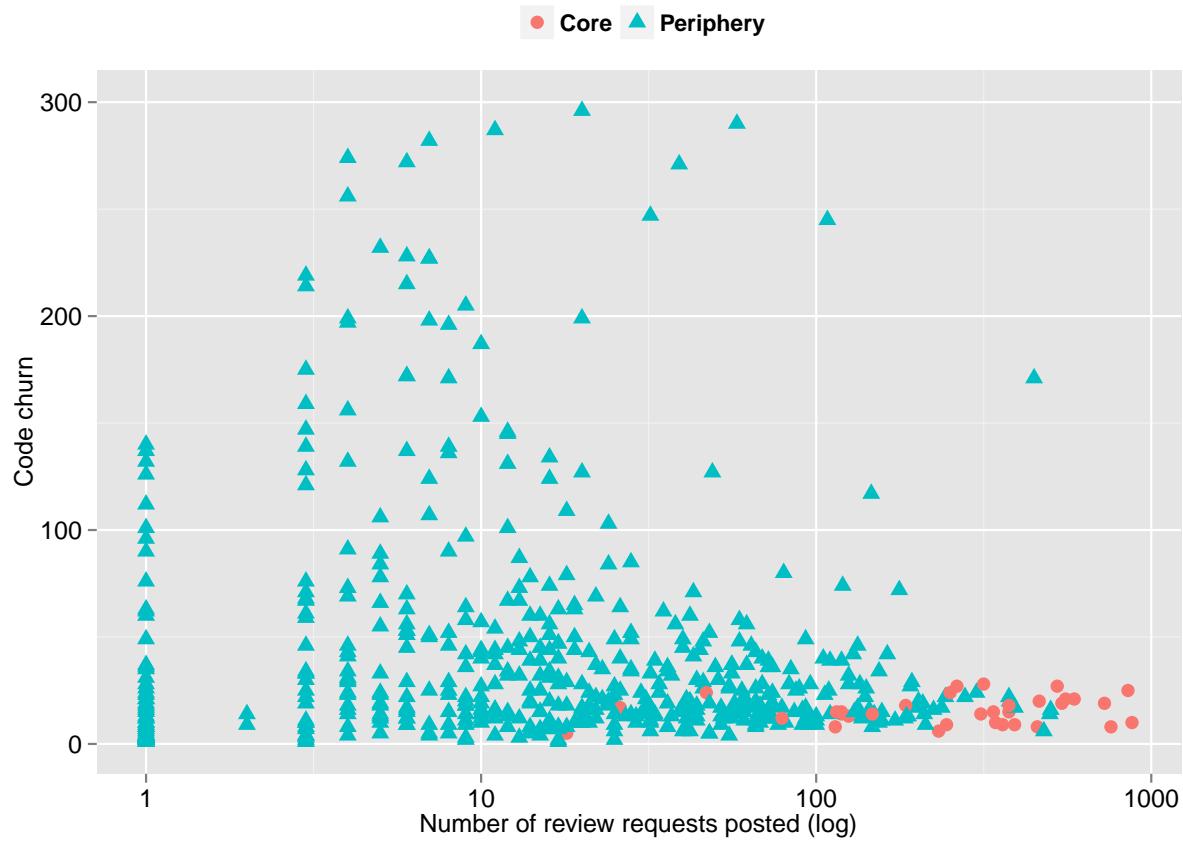


Figure 4.12: Experience vs. Code churn (OmapZoom)

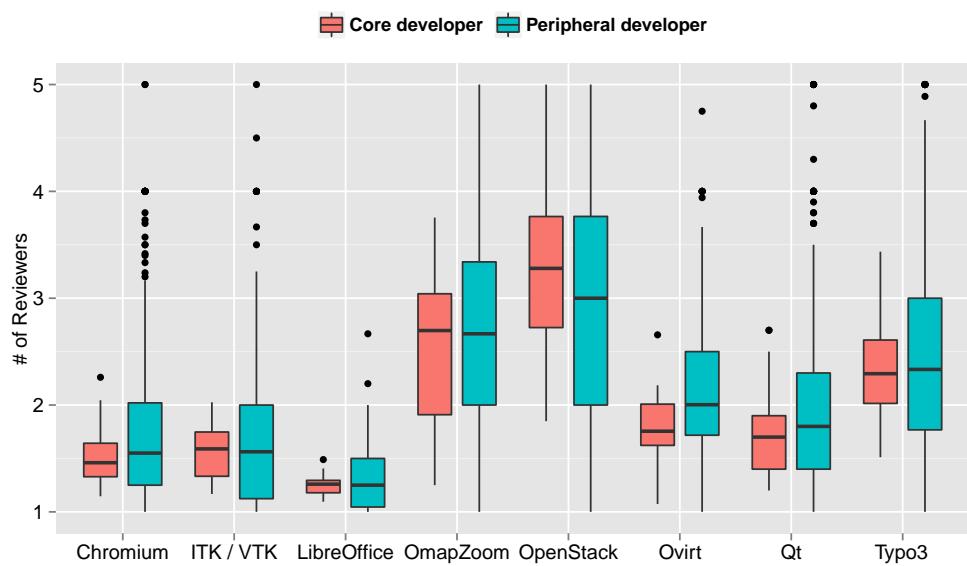


Figure 4.13: Number of reviewers

Table 4.9: Statistical Results for H7: Number of reviewers

Project	Number of reviewers		
	Core	Periphery	
Chromium OS	1.46	1.55	p=.059
ITK/VTK	1.59	1.56	p=.918
LibreOffice	2.08	2.33	p=.983
OmapZoom	2.70	2.72	p=.305
OpenStack	3.30	3.00	p=.284
OVirt	1.75	2.09	p=.006*
Qt Project	1.68	1.80	p=.149
Typo3	2.29	2.38	p=.654

* Indicates a statistically significant difference

Table 4.10: Statistical Results for Group differences

Project	Median First Interval (hours)				Kruskal Wallis W
	C† to C	P‡ to C	C to P	P to P	
Chromium OS	1.02	1.16	2.99	2.75	$\chi^2 = 37.66, p < 0.001*$
ITK/VTK	3.13	3.93	18.32	17.62	$\chi^2 = 18.19, p < 0.001*$
LibreOffice	6.50	13.6	19.7	16.80	$\chi^2 = 19.25, p < 0.001*$
OmapZoom	1.92	0.36	15.81	0.84	$\chi^2 = 138.42, p < 0.001*$
OpenStack	3.19	1.70	6.82	3.12	$\chi^2 = 106.47, p < 0.001*$
OVirt	6.40	4.40	11.60	14.2	$\chi^2 = 9.02, p = 0.03*$
Qt Project	1.83	2.29	6.97	5.06	$\chi^2 = 28.69, p < 0.001*$
Typo3	0.16	0.03	1.17	0.04	$\chi^2 = 72.83, p < 0.001*$

C† = Core developer, P‡ = Peripheral developer

* Indicates a statistically significant difference

4.5.1 Inter/intra group collaboration

Section 4.4 discusses the results based on the position of an author in the core or periphery.

However, we hypothesized that the position of a reviewer may influence his review process. Table 4.2 indicates that, although core developers comprise only 5%-12% of the community, they reviewed 60%-90% code changes. Because core members receive high number of review requests, their review time may be longer than for peripheral developers. In addition, the core developers have to prioritize review requests. Some core developers may prioritize reviewing changes from other core developers, while others may focus on areas that need most attention (i.e., changes from newcomers). This section analyzes whether feedback intervals depends on whether the author and the reviewer belong to the same group (i.e. core - core, core- periphery, periphery - core, and periphery -periphery).

Figure 4.14 shows the first feedback intervals for the eight projects based on the group of

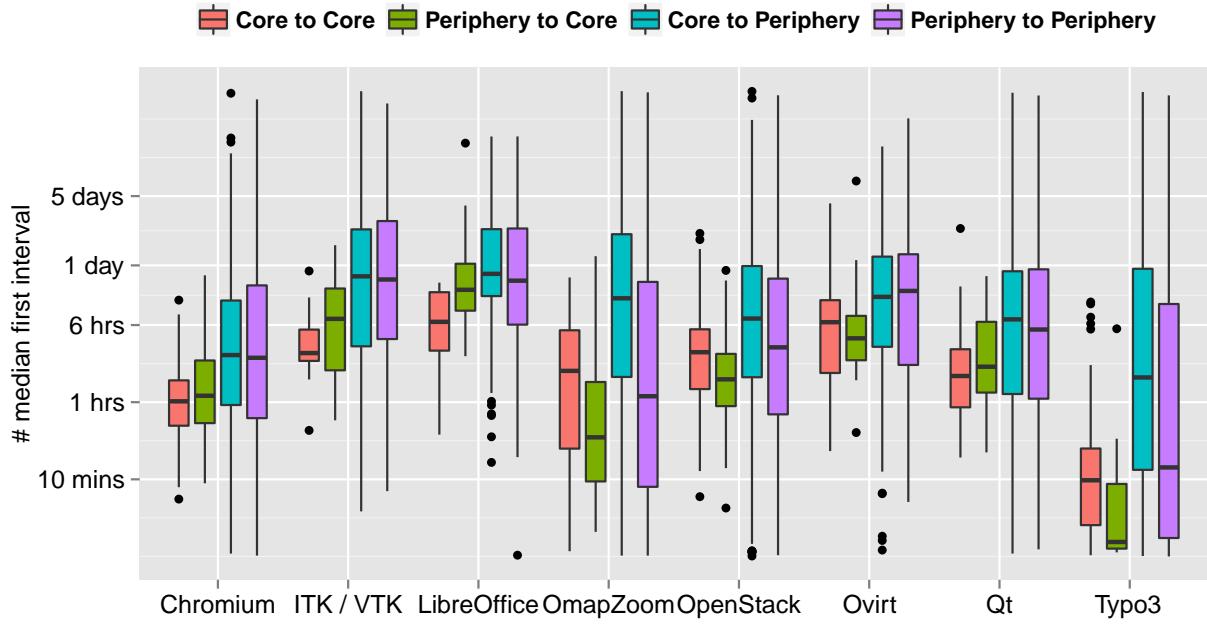


Figure 4.14: Differences in first feedback intervals during inter / intra group collaborations

the author and the reviewer, where Core to Periphery indicates that the reviewer belonged to the core and the author belonged to the periphery. Table 4.10 shows the median first feedback intervals for the four types of communications. The results of non-parametric Kruskal Wallis W test indicate that the differences are significant for all eight projects.

For all projects, the *core to periphery* reviews were the slowest, suggesting that the core developers may give a lower priority to reviewing changes from peripheral developers. Interestingly, in all projects *periphery to core* reviews were significantly faster than the *periphery to periphery* reviews. In fact, *periphery to core* reviewers were the fastest in half of the projects. This result suggests that peripheral developers prioritize core developers over other peripheral developers. A potential source of this result is that a peripheral developer may view the invitation to review code from a core members as an opportunity to learn and to build a relationship with that core developer.

Core to core reviews were the fastest for the other half of the projects, suggesting that core developers prioritize code changes from other core members. The code review social networks show thick edges (i.e. high number of reviews) between many core developers. Since two core members who review each other code very frequently are more likely to be aware of each others' strength and weakness, they are able to expedite each others' code reviews by focusing

more on weak areas. Again a pair of frequently interacting core developers are more likely to have a better relationship, which may encourage them to prioritize each other's review requests. In summary, the inter/intra group collaboration indicates a strong influence of reputation and relationships, as core developers receive faster feedback regardless of the status of the reviewer.

4.5.2 Longitudinal analysis of reputation

The results also showed that, across all 8 projects, those developers at the bottom end of the experience spectrum, i.e. those that have posted less than 10 review requests, have very long first feedback and review intervals and very low acceptance rates. Generally, these developers are newcomers to the project, some of whom may intend to become regular contributors. Based on the results, we could hypothesize that as a contributor gains project experience and recognition by posting more review requests, s/he should begin to receive feedback on their changes more quickly. Because our results only show correlation and not causation, we must conduct further analysis to determine whether causation is present.

To gather some evidence for causation, we identified participants who started as newcomers and moved into the *core* through making a large number of contributions. Due to the wide variation in review intervals across projects, making cross-project comparisons would be difficult. We chose to focus our analysis on the Qt project because it had the most review requests of any project in this study. We identified 63 core developer in Qt as of January 2014. Of those, 59 developers belonged to the core during March 2012 (using 10 months' code review data). Only four developers first begun contributing to Qt after March 2012 and moved into the core through their contributions. Figure 4.15 shows the median first feedback intervals, median first review intervals and average acceptance rate for those four developers over project experience. All four developers had a long first feedback and review interval at the beginning. After posting 10-25 review requests, their first feedback and review intervals were more in line with the other *core* developers. This trend seems to provide evidence for the conclusion that as developer gain reputation and experience (i.e. by posting more review requests) their requests receive quicker feedback and they complete the review process quicker.

Conversely, when examining acceptance rate, these developers do not fit the expected pattern established by the results of H3. Three of the four developers had an acceptance rate

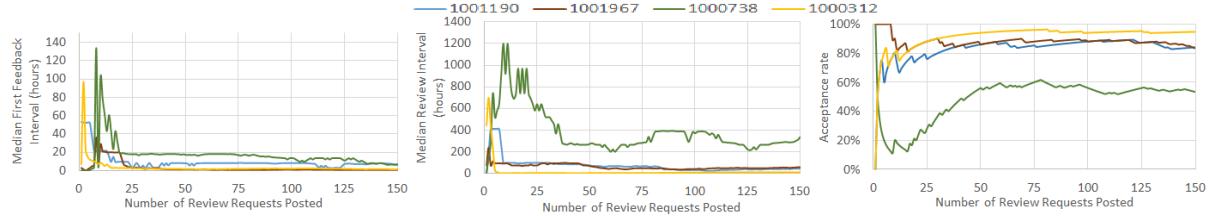


Figure 4.15: Developers moving from periphery to core while starting as newcomer a) running median first feedback interval (left), b) running median review interval (middle), 3) running average acceptance rate (right)

greater than 80% for their first 10 review requests. However, most of the prospective OSS joiners are not successful (Ducheneaut, 2005), and even may not be able to get past 10 review submissions. Together, these two points suggest that a high initial acceptance rate may be an indicator that a newcomer has the potential to move into the core. More analysis is required to draw any conclusion about this observation.

4.6 Threats to Validity

This section is organized around four common threats to validity.

4.6.1 Internal validity

The primary internal validity threat is project selection. While we only included projects that use modern code review supported by Gerrit, we included the majority of the publicly accessible Gerrit-based projects that contain a large number of code review requests. Furthermore, it is possible that projects supported by other code review tools could have behaved differently. We think this threat is minimal for three reasons: 1) all code review tools support the same basic purpose, i.e. detecting defects and improving the code, 2) the basic workflow (i.e. authors posting code, reviewers commenting about code snippets, and code requiring approval from reviewer before integration) of most of the code review tools are similar 3) we did not use any Gerrit-specific feature/attributes in this study. Therefore, we believe the project selection threat is minimal.

4.6.2 Construct validity

The primary threat to construct validity is regarding our approach to partition an OSS community into core and periphery groups. It is possible that our approach placed some developers

in the wrong group. However, the validation of the CIK approach (Bosu and Carver, 2014b) increases our confidence that the number misgrouped developers is small. Because each project had a large number of developers, a few misgrouped developers will not significantly alter the results.

Second, we assume that the code review outcomes measured in this study (i.e. first feedback time, review interval, acceptance rate, and number of patchsets per review request) are influenced by a developer’s position in the code review social network. However, there may be number of unmeasured confounding factors that could influence a review outcome (e.g., availability of developers, code complexity, status of the author as a volunteer or a paid contributor, and licensing terms). There is no evidence that our partitioning approach or our measurements would be systematically biased based on any of these confounding factors. Furthermore, the fact that the results were similar across all eight projects provides additional confidence that any confounding factors did not significantly impact the study results.

Finally, the assumption that core developers have more recognition than the peripheral developers is a more minor threat. While this assumption is likely true for most core developers, there may be few exceptions. As an extreme example, if an iconic OSS figure (e.g. Linus Torvalds, or Guido Van Rossum) or another well-known developer makes occasional contributions to an OSS project, he would be classified as a peripheral developer but would have more recognition in the community than the many of the core members. Because the projects used in this study had a large number of developers, any exceptions like this would be minimal and would not impact the overall results.

4.6.3 External validity

The OSS projects in this study vary across domains, languages, age, and governance. In addition, we analyzed a large number of code review requests for each project. Therefore, we believe these results can be generalized to many other OSS projects. However, because of the wide variety of OSS project characteristics, we do not claim that these results are universal for all OSS projects. Drawing a more general conclusion would require a family of experiments (Basili et al., 1999) that included OSS projects of all types. To encourage this type of

replication, we have made our scripts available to other interested researchers⁴.

4.6.4 Conclusion validity

The use of large dataset drawn from eight projects, which produced similar result across those eight different projects, boosts confidence about the validity of the results. We tested all data for normality prior to conducting statistical analyses and used appropriate tests based upon the results of the normality test. We used popular and well-known social network analysis tools for this study. Therefore, threats to conclusion validity are minimal.

4.7 Conclusion

To identify the impact of an OSS developer's reputation on the outcome of his/her contributions, we performed a social network analysis of the code review data from eight popular OSS projects based on the assumption that core developers have a better reputation within the community than the peripheral developers. The results indicate that the *core* developers receive their first feedback more quickly and get a higher percentage of their changes accepted than the *peripheral* developers. Even though the *core* developers do not require fewer patchset submissions than the *peripheral* developers, they are able to get their code accepted more quickly. Taken together, these results suggest that *core* developers may also receive quicker subsequent feedback, although we did not test this hypothesis. This result is not surprising, *core* developers should enjoy some benefits, like quicker feedback, because they have built a reputation over time. However, if the feedback intervals are 2 to 19 times (or 12 to 96 hours) longer for the *peripheral* developers than for the *core* developers, those *peripheral* developers may lose the motivation to participate in the project. Conversely, if those peripheral developers received quicker feedback, they may be more motivated to contribute to the project. Because the vast majority of OSS participants are *peripheral* developers (>80%), any measure that could increase their participation has the potential to have a great impact on the project.

Based on the results of this study, we hypothesize that OSS projects could benefit by developing a triage process to help newcomers receive quicker feedback. Even though it may not be possible to increase the acceptance rate for newcomers (because it could compromise

⁴ <http://asbosu.students.cs.ua.edu/sna-code-review/index.html>

project quality), OSS projects can encourage newcomers by providing quicker feedback on review requests. If a newcomer has to wait too long for feedback, s/he may feel ignored, lose interest, and ultimately leave the project. This attrition may have a negative effect on an OSS project. Research suggests that projects that fail to attract and, more importantly, to retain new contributors cannot survive (Ducheneaut, 2005). Conversely, if a newcomer receives feedback quickly, s/he is more likely to make the requested changes so his/her code can be merged into the project code base. Having his/her changes accepted will help a newcomer become recognized and build reputation within the project (Raymond, 1998). This recognition will, in turn, motivate him/her to contribute more (Gacek and Arief, 2004) and increase the overall project output.

The result of H6, which indicates that peripheral developers submit larger patchsets than the core developers, is surprising. The result also showed that many of the newcomers commit large patchsets, which goes against the suggestions to newcomers from prior research (Von Krogh et al., 2003; Herraiz et al., 2006; Ducheneaut, 2005) to submit small patches to fix bugs. Newcomers' negligence of those suggestions not only reduces their acceptance rate but also increases review time, and discourages them from future contributions. OSS projects should provide some guidelines for newcomers and suggest the areas they should work on. For example, the Python developer's guide⁵ includes the areas a newcomer should look into to become familiar with the project and the process, before they can start working on large patchsets. We believe similar type of guide can be helpful for other projects to successfully train newcomers.

This study has made several contributions. The first key contribution of this study is the empirical evidence regarding the effect of OSS developers' reputation on code review outcome. The lack of an established reputation may result in longer delays to complete the review process. Moreover, we can conclude that a developer's reputation has the most impact when s/he is a newcomer. However, once s/he has build some recognition by posting a minimum number of review requests, which we found to be between 10-25, s/he may enjoy shorter feedback intervals, which may be similar to the core developers.

Another key contribution of this study is our novel "Core Identification using K-means

⁵ <https://docs.python.org/devguide/>

(CIK)” approach. There has not been any existing approach to identify core nodes in large multi-core networks (e.g., most of the popular OSS communities). We believe our approach will be useful for other researches to study the core nodes in large multi-core networks.

The results of the study suggest two key research directions: 1) building a reviewer recommendation system to triage incoming review requests to enable faster feedback, and 2) a better understanding of OSS newcomers’ submission of large patchsets as well as failure to continue collaborations which can enable the OSS projects to the lower attrition of prospective newcomers.

4.8 References

- Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721.
- Basili, V. R., Shull, F., and Lanubile, F. (1999). Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473.
- Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. In *Proceedings of the 3rd International AAAI Conference on Weblogs and Social Media*.
- Bird, C., Gourley, A., Devanbu, P., Gertz, M., and Swaminathan, A. (2006). Mining email social networks. In *Proceedings of the 3rd International Workshop on Mining Software Repositories*, pages 137–143.
- Bird, C., Pattison, D., D’Souza, R., Filkov, V., and Devanbu, P. (2008). Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 24–35.
- Bonacich, P. (2007). Some unique properties of eigenvector centrality. *Social Networks*, 29(4):555–564.

- Borgatti, S. P. and Everett, M. G. (2000). Models of core/periphery structures. *Social networks*, 21(4):375–395.
- Bosu, A. and Carver, J. C. (2012). Peer code review in open source communities using review-board. In *Proceedings of the 4th ACM Workshop on Evaluation and Usability of Programming Language and Tools*, pages 17–24.
- Bosu, A. and Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. In *Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 133–142.
- Bosu, A. and Carver, J. C. (2014a). How do social interaction networks influence peer impressions formation? a case study. In *Open Source Software: Mobile Open Source Technologies*, volume 427 of *IFIP Advances in Information and Communication Technology*, pages 31–40.
- Bosu, A. and Carver, J. C. (2014b). Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *Proceedings of the 2014 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 33:1–33:10.
- Crowston, K. and Howison, J. (2005). The social structure of free and open source software development. *First Monday*, 10(2-7).
- Dinh-Trong, T. and Bieman, J. (2005). The freebsd project: a replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494.
- Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368.
- Everett, M. G. and Borgatti, S. P. (2000). Peripheries of cohesive subsets. *Social Networks*, 21(4):397–407.
- Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182 –211.
- Fielding, R. T. (1999). Shared leadership in the apache project. *Communications of the ACM*, 42(4):42–43.

- Freeman, L. (2004). *The development of social network analysis: A study in the sociology of science*, volume 1. Empirical Press Vancouver.
- Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41.
- Gacek, C. and Arief, B. (2004). The many meanings of open source. *Software, IEEE*, 21(1):34–40.
- Hann, I.-H., Roberts, J., and Slaughter, S. (2004). Why developers participate in open source software projects: An empirical investigation. *Proceedings of the ICIS 2004*, page 66.
- Herraiz, I., Robles, G., Amor, J. J., Romera, T., and González Barahona, J. M. (2006). The processes of joining in global distributed software projects. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33.
- Kampstra, P. et al. (2008). Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of Statistical Software*, 28(1):1–9.
- Lakhani, K. (2006). *The Core and Periphery in Distributed and Self-organizing Systems*. PhD thesis, Doctoral Dissertation, Sloan School of Management, MIT.
- Lakhani, K. R. and Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on free and open source software*, 1:3–22.
- Lee, G. K. and Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the linux kernel development. *Organization science*, 14(6):633–649.
- Long, Y. and Siau, K. (2007). Social network structures in open source software development teams. *Journal of Database Management (JDM)*, 18(2):25–40.
- MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14.

- Markus, M. L., Manville, B., and Agres, E. (2000). What makes a virtual organization work? *Sloan Management Review*, 42(1):13–26.
- Martinez-Romo, J., Robles, G., Gonzalez-Barahona, J. M., and Ortuño-Perez, M. (2008). Using social network analysis techniques to study collaboration between a FLOSS community and a company. In *Open Source Development, Communities and Quality*, pages 171–186. Springer.
- Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering Methodology*, 11(3):309–346.
- Mukadam, M., Bird, C., and Rigby, P. C. (2013). Gerrit software code review data from android. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 45–48.
- Newman, M. (2010). *Networks: An introduction*. Oxford University Press.
- Oezbek, C., Prechelt, L., and Thiel, F. (2010). The onion has cancer: Some social network analysis visualizations of open source project communication. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Dev.*, pages 5–10.
- Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web.
- Raymond, E. S. (1998). Homesteading the noosphere. *First Monday*, 3(10).
- Rigby, P. C. and Bird, C. (2013). Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 202–212.
- Rigby, P. C., German, D. M., and Storey, M.-A. (2008). Open source software peer review practices: a case study of the Apache server. In *Proceedings of the 30th International Conference on Software Engineering, ICSE '08*, pages 541–550.

- Robles, G., Gonzalez-Barahona, J. M., and Herraiz, I. (2009). Evolution of the core team of developers in libre software projects. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR'09.*, pages 167–170.
- Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, 31(4):581–603.
- Terceiro, A., Rios, L. R., and Chavez, C. (2010). An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *Proceedings of the 2010 Brazilian Symposium on Software Engineering (SBES)*, pages 21–29.
- Von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241.
- Wasserman, S. (1994). *Social network analysis: Methods and applications*, volume 8. Cambridge university press.
- Ye, Y. and Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering, ICSE 2003.*, pages 419–429.

CHAPTER 5

OVERALL CONCLUSION

The primary goal of this dissertation was to:

Better understand contemporary code review practices, its benefits, and factors that influence its outcomes.

Based on this high level goal, Section 1.2 identified six specific research objective for this dissertation. To fulfill those objectives, I surveyed the code review participants from popular OSS projects and Microsoft. The results of those two surveys provide insights for achieving five of the research objectives (i.e., RO1, RO2, RO3, RO5, RO6). Then, I analyzed social interaction network of eight open source projects to understand the impact of developers' reputation on code review outcomes. The results of that analysis provide insights for two research objectives (i.e., RO4, and RO6).

Based on these results, this dissertation provided a better understanding of contemporary code reviews in general, the technical and non-technical benefits of code reviews, and the factors that influence the outcomes of code review.

The remainder of this chapter is organized as follows. Section 5.1 summarizes the outcome for the six objectives defined in Section 1.2. Section 5.2 summarizes the contributions for this dissertation. Finally, Section 5.3 provides directions for future research.

5.1 Key Outcomes

The following subsections summarize the key outcomes relative to the six research objectives of this dissertation.

5.1.1 RO1: *Better understand developers' rationale behind practicing code reviews.*

Although, more than three-fourth of the code review comments do not find functionality issues (Bosu et al., 2015; Beller et al., 2014), developers sill consider code reviews very impor-

tant for their projects for reasons including: knowledge dissemination, relationship building, better designs, and ensuring maintainable code. For a large scale and or long term projects, those benefits may be very important and hard to achieve through other ways. Therefore, even projects with highly-skilled developers who rarely commit low-quality changes can still benefit from the practice of code review.

5.1.2 RO2: *Better understand contemporary code review process.*

The following list summarizes the key insights from this dissertation to better understand contemporary code review process.

1. On average, developers spend 10-15% of their time in code reviews. Moreover, the amount of efforts spent in code reviews usually increases with development experience.
2. Prior history of interactions with and reputation of the code author influences not only the decision to accept or reject an incoming review request but also the rigor to review the code change.
3. A code change that is simple, easy to understand, self documenting and requires minimum review time is highly appreciated by the reviewers during code reviews.
4. Reviewers prefer providing comments to help authors fix poor code changes. However, code review comments should be carefully articulated so that those do not hurt the feelings of an author.

These insights not only signify the amount of effort spent in code reviews but also provide a better understandings of decisions during code reviews. These insights can be helpful for project management to improve their code review process.

5.1.3 RO3: *Study the differences between code review practices in OSS and commercial projects.*

The results of a prior study suggest that many of the quantitative metrics (e.g., response time, review interval, number of iterations, patchset size, and number of comments) to characterize the code review process of OSS and commercial projects were largely similar (Rigby and

Bird, 2013). Conversely, this dissertation primarily focused on two qualitative aspects (i.e., decisions, and impressions formation) of code review and found significant differences between the opinions of the OSS and Microsoft developers. The focus for OSS reviewers is on building relationships with core team members. When forming impressions of their teammates through code reviews, the OSS respondents indicated that the personal characteristics and work habits of the code author were the most important factors. Conversely, the Microsoft respondents found identification of the author's expertise and of the potential for future collaborations to be the most important factors.

In addition, when accepting or rejecting a review request, OSS reviewers focused more on their relationship with and reputation of the code author. Conversely, the expertise of the author (i.e., if a developer will be able to provide useful reviews, if a developer writes good code that he can learn, and if he has expertise to review that code) and time/effort to review a code change were primary considerations for the Microsoft reviewers.

5.1.4 RO4: Better understand the characteristics of code review social networks.

The results indicate that code review networks of OSS projects are highly centralized and largely dominated by a small portion (between 5-10%) of the participants. Usually those participants are the core developers of the OSS project, who perform more than 60-80% of code reviews. Moreover, core developers are also responsible for the majority of the code changes. Prior research suggests that an OSS contributor can become a core developer only through quality contributions for a long period of time. However, only a small number of prospective joiners of OSS projects were successful to become core members.

5.1.5 RO5: Identify various non-technical benefits of code reviews.

While there is empirical evidence that code review improves software quality (Wiegers, 2002; Cohen et al., 2006), the evidence about the non-technical benefits has been mostly anecdotal. One of the key non-technical benefits of code review is its role in impression formation, that is how developers form opinions of teammates. This dissertation not only provides empirical evidence regarding the impact of code review on peer impressions formation but also provides insight on how those impressions form. The quality of the code submitted for review is an important aspect of the formation of teammate perceptions. Moreover, the impressions

formed during code review could also affect future collaborations, relationships, and work practices. Therefore, the non-technical benefits of code reviews are crucial for forming the social underpinning of successful projects.

5.1.6 RO6: *Identify the human factors that influence code review process and it's outcomes.*

This dissertation reports empirical evidence regarding the impact of four human factors (i.e., reputation, trust, perception of expertise, and relationship) on the code review process and its outcomes. The following paragraphs summarize those findings.

Reputation: Reputed developers of OSS projects receive quicker feedback for review requests, are able to complete code reviews faster, and are able to get a larger percentage of code changes accepted than the less reputed developers.

Trust: Reviewers are more likely to perform more thorough reviews of code submitted by teammates who are untrustworthy than of code submitted by teammates they view as experts.

Perception of expertise: The effect of perceived expertise of the author is mixed. Some reviewers preferred to review code from experts to learn as well as minimize their time spent in reviews. Other reviewers preferred prioritized reviews from newcomers or focused on area requiring most attentions.

Relationship: Many reviewers, especially those contributing to OSS projects, emphasize factors related to their relationship with and history of interaction with the code author when accepting an incoming review request.

5.2 Key Contributions

The key contributions of this dissertation include:

- Empirical insights into the peer impression formations in OSS organizations. These insights can be helpful to leverage software development practices to support team-building.
- Better understanding of developers' perceived benefits of contemporary code reviews. These insights can help project managers decide whether code reviews can be useful for their projects.

- The first study to provide empirical evidence regarding impact of contemporary code review on peer impressions formation. These additional benefits may further motivate project managers to adopt code reviews.
- Empirical evidence regarding various technical and non-technical benefits of code reviews. The evidence about the several key benefits of code review (e.g. sharing knowledge, sharing expertise, sharing development techniques, and most importantly forming peer impressions) has been mostly anecdotal. This dissertation bridged that research gap.
- Better understanding of why and how developers collaborate during code reviews. These insights can be helpful for practitioners to improve their code review process.
- Empirical evidence regarding the effect of an OSS developer's reputation on the outcome of his/her code review requests. These insights identified improvement areas for OSS code review process and suggested future research directions.
- Core Identification using K-means (CIK), a novel approach to group OSS developers into core and periphery. There has not been any existing approach to identify core nodes in large multi-core networks (e.g., most of the popular OSS communities). This approach can be useful for other researches to study the core nodes in large multi-core networks.
- Illustration of systematically designing and analyzing a software engineering (SE) survey. The example from this dissertation can help researchers design future SE surveys.

5.3 Directions for Future Work

This dissertation identified several improvement opportunities to improve contemporary code reviews that requires attention from researchers. First, results of this dissertation indicate that newcomers of OSS projects has to wait 2-12 times longer to receive first feedback for their review requests. If a newcomer has to wait too long for feedback, s/he may feel ignored, lose interest, and ultimately leave the project. This attrition may have a negative effect on an OSS project. Research suggests that projects that fail to attract and, more importantly, to retain new contributors cannot survive (Ducheneaut, 2005). Therefore, OSS projects should benefit by

developing a triage process to help newcomers receive quicker feedback. Researchers should develop methods to support this need.

Second, most of the code review comments are about minor stylistic issues (Bosu et al., 2015; Beller et al., 2014). There are static analysis tools (e.g., OACR, FxCop and StyleCop) that are able to identify many of those stylistic issues. There is a need for integrating those tools with existing code review tools like Gerrit, that can provide automated review comments for those style issues. This integration can reduce review time since human reviewers can focus more on identifying functional defects or design issues. One possible drawback is that the static analysis tools report a wide range of issues. Many of those may not require attention from the author. If the ratio of such review comments are very high, then it may be annoying to an author. Therefore, researchers need to determine the ways to customize those tools, so they can be integrated as automated reviewers.

Finally, the results of this dissertation suggest empirical evidence regarding the impact of code reviews on impressions formation between developers. However, I hypothesize that they way a review comment is articulated also influences impression formation. For example, if an author is offended by the comments of a reviewer, s/he might try to avoid that reviewer in future collaborations. Conversely, developers are more likely to collaborate with peers who are helpful and articulate.

Moreover, I hypothesize that the articulation of a review comment also influence its outcomes. For example, if a reviewer is rude while asking for a change, the author might argue against it. Conversely, if a reviewer provides constructive criticism, the author might consider that with positive intentions.

However, there is no empirical evidence supporting those hypotheses regarding the impact of articulation of comments on impressions formation as well as code review outcomes. Therefore, there is a need to study these two hypotheses.

5.4 Publications

Thus far, I have published one journal paper, six peer reviewed conference papers, four peer reviewed short papers, and two other papers (workshop / doctoral symposium). I have

two journal papers complete (pending submission). Below is the list of the papers in reverse chronological order.

5.4.1 Journal Papers

- Published
 1. **Amiangshu Bosu**, Jeffrey C. Carver, Rosanna Guadagno, Blake Bassett, Debra McCallum, and Lorin Hochstein. “Peer impressions in open source organizations: A survey.” *Journal of Systems and Software*, 94(0):4 – 15, 2014.
- In Preparation
 1. **Amiangshu Bosu**, Jeffrey C. Carver, and Christian Bird. “Insights into the Process Aspects and Social Dynamics of Contemporary Code Review” Under preparation for *IEEE Transactions on Software Engineering*, Submission date: June 25, 2014 (tentative).
 2. **Amiangshu Bosu**, Jeffrey C. Carver “Developer Collaboration in OSS Projects Through Code Review.” Under preparation for *Empirical Software Engineering Journal*, Submission date: July 15, 2015 (tentative).

5.4.2 Refereed Conference Papers

- Published
 1. **Amiangshu Bosu**, Michaela Greiler, and Christian Bird. “Characteristics of Useful Code Reviews: An Empirical Study at Microsoft.” In *Proceedings of the 12nd Working Conference of Mining Software Repositories (MSR)*, To appear, Florence, Italy, 2015.
 2. **Amiangshu Bosu**, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. “Identifying the characteristics of vulnerable code changes: An empirical study.” In *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE*, pages 257-268, Hong Kong, China, 2014.

3. **Amiangshu Bosu** and Jeffrey C. Carver. “Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation.” In *Proceedings of the 2014 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 33:1–33:10, Torino, Italy, 2014.
4. **Amiangshu Bosu** and Jeffrey C. Carver. “How do social interaction networks influence peer impressions formation? A case study.” In Luis Corral, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, and AnthonyI. Wasserman, editors, *Open Source Software: Mobile Open Source Technologies*, volume 427 of *IFIP Advances in Information and Communication Technology*, pages 31–40. Springer Berlin Heidelberg, 2014 (*Proceedings on the 10th International Conference on Open Source Systems published as a book*).
5. **Amiangshu Bosu** and Jeffrey C. Carver. “Impact of peer code review on peer impression formation: A survey.” In *Proceedings of the 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 133–142, Baltimore, MD, USA, 2013.
6. **Amiangshu Bosu**, Christopher S. Corley, Dustin Heaton, Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. “Building reputation in StackOverflow: An empirical investigation.” In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 89–92, San Francisco, CA, USA, 2013.

5.4.3 Doctoral Symposium / Workshop Papers

- Published

1. **Amiangshu Bosu**. “Modeling modern code review practices in open source software development organizations.” In *Proceedings of the 11th International Doctoral Symposium on Empirical Software Engineering*, Baltimore, MD, USA, 2013.
2. **Amiangshu Bosu** and Jeffrey C. Carver. “Peer code review in open source communities using ReviewBoard.” In *Proceedings of the ACM 4th annual workshop on Evaluation and usability of programming languages and tools (PLATEAU)*, pages 17–24, Tucson, Arizona, USA, 2012.

5.4.4 Refereed Short Papers

- Published
1. **Amiangshu Bosu.** “Characteristics of the vulnerable code changes identified through peer code review.” In *Companion Proceedings of the 36th International Conference on Software Engineering- Student Research Competition Track*, Hyderabad, India , 2014, pages 736–738.
 2. **Amiangshu Bosu**, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. “When are OSS developers more likely to introduce vulnerable code changes? A case study.” In Luis Corral, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, and AnthonyI. Wasserman, editors, *Open Source Software: Mobile Open Source Technologies*, volume 427 of *IFIP Advances in Information and Communication Technology*, pages 234–236. Springer Berlin Heidelberg, 2014.
 3. **Amiangshu Bosu** and Jeffrey C. Carver. “Peer code review to prevent security vulnerabilities: An empirical evaluation.” In *Proceedings of the 2013 IEEE 7th International Conference on Software Security and Reliability-Companion (SERE-C)*, pages 229–230, Washington, D.C., USA, 2013.
 4. **Amiangshu Bosu.** “Mining repositories to reveal the community structures of open source software projects.” In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE ’12, pages 397–398, Tuscaloosa, Alabama, 2012.

REFERENCES

- Asundi, J. and Jayant, R. (2007). Patch review processes in open source software development communities: A comparative case study. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, 2007. HICSS 2007.*, pages 166c–166c.
- Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721.
- Balachandran, V. (2013). Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 931–940.
- Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2013). The influence of non-technical factors on code review. In *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 122–131.
- Beller, M., Bacchelli, A., Zaidman, A., and Juergens, E. (2014). Modern code reviews in open-source projects: which problems do they fix? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 202–211.
- Bosu, A., Carver, J., Guadagno, R., Bassett, B., McCallum, D., and Hochstein, L. (2014). Peer impressions in open source organizations: A survey. *Journal of Systems and Software*, 94(0):4 – 15.
- Bosu, A. and Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. In *Proceedings of the 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 133–142.
- Bosu, A. and Carver, J. C. (2014). Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *Proceedings of the 2014 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 33:1–33:10.

Bosu, A., Greiler, M., and Bird, C. (2015). Characteristics of useful code reviews: An empirical study. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, page TBD.

Cohen, J., Brown, E., DuRette, B., and Teleki, S. (2006). *Best Kept Secrets of Peer Code Review*. Smart Bear.

Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15:182–211.

Guadagno, R. and Cialdini, R. (2005). Online persuasion and compliance: social influence on the internet and beyond. *The Social Net: Human Behavior in Cyberspace*, pages 91–113.

Johnson, P. M. (1998). Reengineering inspection. *Communications of the ACM*, 41(2):49–52.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 192–201.

Raymond, E. S. (1998). Homesteading the noosphere. *First Monday*, 3(10).

Rigby, P., Cleary, B., Painchaud, F., Storey, M.-A., and German, D. (2012). Contemporary peer review in action: Lessons from open source development. *IEEE Software*, 29(6):56–61.

Rigby, P. C. and Bird, C. (2013). Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ES-EC/FSE 2013, pages 202–212.

Rigby, P. C. and German, D. M. (2006). A preliminary examination of code review processes in open source projects. Technical Report DCS-305-IR, University of Victoria.

Rigby, P. C., German, D. M., and Storey, M.-A. (2008). Open source software peer review practices: a case study of the apache server. In *Proceedings of the 30th International Conference on Software Engineering*, pages 541–550.

Rigby, P. C. and Storey, M.-A. (2011). Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE ’11, pages 541–550.

Sutherland, A. and Venolia, G. (2009). Can peer code reviews be exploited for later information needs? In *Proceedigns of the 31st International Conference on Software Engineering Companion Volume, ICSE-Companion 2009.*, pages 259–262.

Votta Jr, L. G. (1993). Does every inspection need a meeting? In *ACM SIGSOFT Software Engineering Notes*, volume 18, pages 107–114.

Wiegers, K. E. (2002). *Peer reviews in Software: A practical guide*. Addison-Wesley Boston.

Wohlin, C., Höst, M., and Henningsson, K. (2003). Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer.

APPENDIX A
APPENDICES

March 30, 2011

Office for Research

Institutional Review Board for the
Protection of Human Subjects



Jeffrey C. Carver, Ph.D.
Department of Computer Science
College of Engineering
The University of Alabama

Re: IRB # 11-OR-103 "Survey of Open Source Developers"

Dear Dr. Carver:

The University of Alabama Institutional Review Board has granted approval for your proposed research

Your application has been given expedited approval according to 45 CFR part 46. You have also been granted the requested waiver of written documentation of informed consent. Approval has been given under expedited review category 7 as outlined below:

(7) Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.

Your application will expire on March 29, 2012. If your research will continue beyond this date, complete the relevant portions of Continuing Review and Closure Form. If you wish to modify the application, complete the Modification of an Approved Protocol Form. When the study closes, complete the appropriate portions of FORM: Continuing Review and Closure.

Please use reproductions of the IRB approved informed consent form to obtain consent from your participants.

Should you need to submit any further correspondence regarding this proposal, please include the above application number.

Good luck with your research.

Sincerely,



Carpantalo T. Myles, MSM, CIM
Director & Research Compliance Officer
Office for Research Compliance
The University of Alabama

IRB Project #: 11-OR-103

UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS

I. Identifying information

	Principal Investigator	Second Investigator	Third Investigator
Names:	Jeffrey Carver	Debra McCallum	Rosanna E. Guadagno
Department:	Computer Science	SSRC	Psychology
College:	Engineering	Arts & Sciences	Arts & Science
University:	Alabama	Alabama	Alabama
Address:	Box 870290	391A Nott Hall	Box 870348
Telephone:	8-9829	8-3820	8-7803
FAX:	8-0219		8-8648
E-mail:	carver@cs.ua.edu	dmccallu@as.ua.edu	rosanna@ua.edu

Title of Research Project: Survey of Open Source Developers

Date Submitted: February 22, 2011

Funding Source: N/A

Type of Proposal	<input checked="" type="checkbox"/> New	<input type="checkbox"/> Revision	<input type="checkbox"/> Renewal Please attach a renewal application	<input type="checkbox"/> Completed	<input type="checkbox"/> Exempt
Please attach a continuing review of studies form					
Please enter the original IRB # at the top of the page					

UA faculty or staff member signature: [REDACTED]

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):

Type of Review: Full board Expedited

IRB Action:

- Rejected Date: _____
 Tabled Pending Revisions Date: _____
 Approved Pending Revisions Date: _____

Approved-this proposal complies with University and federal regulations for the protection of human subjects.

Approval is effective until the following date: 3-29-12

- Items approved: Research protocol (dated _____)
 Informed consent (dated _____)
 Recruitment materials (dated _____)
 Other (dated _____)

Approval signature [REDACTED]

Date 3/30/2011

**UNIVERSITY OF ALABAMA
Informed Consent for a Research Study**

You are being asked to take part in a research study. This study is called *Survey of Open-Source Software Developers*. The study is being done by *Drs. Jeffrey Carver, Rosanna Guadagno, and Debra McCallum, University of Alabama, and Dr. Lorin Hochstein, University of Southern California*.

What is this study about?

This survey will help us understand your interactions with your teammates.

Why is this study important--What good will the results do?

This study will help us better understand how open-source software developers work together in a distributed environment.

Why have I been asked to take part in this study?

You have been asked to be in this study because you are subscribed to a relevant mailing list.

How many people besides me will be in this study?

Approximately 100 people will participate in the study.

What will I be asked to do in this study?

If you decide to be in this study, you will be asked to complete a survey.

How much time will I spend being in this study?

Approximately 15 minutes.

Will I be paid for being in this study?

No.

Will being in this study cost me anything?

There will be no cost to you.

Can the researcher take me out of this study?

The researcher may take you out of the study if your survey is incomplete.

What are the benefits (good things) that may happen to me if I am in this study?

There are no direct benefits to you from being in this study.

What are the benefits to scientists or society?

This study will help the researchers understand how developers interact in virtual teams.

What are the risks (dangers or harm) to me if I am in this study?

There are no risks to participating in this study. All information will be anonymous.

How will my confidentiality (privacy) be protected? What will happen to the information the study keeps on me?

The surveys will be anonymous.

What are the alternatives to being in this study? Do I have other choices?

The alternative/other choice is not to participate.

What are my rights as a participant?

Taking part in this study is voluntary. You may choose not to take part at all. If you start the study, you can stop at any time. Leaving the study will not result in any penalty or loss of any benefits you would otherwise receive.

Who do I call if I have questions or problems?

If you have questions about the study, please contact Dr. Carver at (205)-348-9828 or carver@cs.ua.edu. If you have any questions about your rights as a research participant you may contact Ms. Tanta Myles, The University of Alabama Research Compliance Officer, at (205)-348-8461 or 1-877-820-3066.

If you have read this consent form, had the study adequately explained to you, understand what you are being asked to do, and freely agree to take part in the survey, click the "next" button. Otherwise, please exit the survey. You should print a copy of this survey for your records.

UA IRB Approved Document
Approval date: 3-30-11
Expiration date: 3-29-12

Email Solicitation:

Subject: Participation Requested: Survey about Open-Source Software Development

Drs. Jeffrey Carver, Rosanna Guadagno, and Debra McCallum, University of Alabama, and Dr. Lorin Hochstein, University of Southern California, are conducting a survey of open-source software developers. This survey seeks to understand how developers on distributed, virtual teams interact with each other. You must be at least 19 years of age to complete the survey. The survey should take approximately 15 minutes to complete.

This survey has been approved by The University of Alabama IRB board.

UA IRB Approved Document
Approval date: 3-30-11
Expiration date: 3-29-12

Office for Research

Institutional Review Board for the
Protection of Human Subjects

THE UNIVERSITY OF
ALABAMA
R E S E A R C H

March 29, 2012

Jeffrey Carver, PhD
Dept. of Computer Science
College of Engineering
Box 870290

Re: IRB#: 11-OR-103-R1 "Survey of Open Source Developers"

Dear Dr. Carver:

The University of Alabama Institutional Review Board has granted approval for your renewal application.

Your protocol has been given expedited approval according to 45 CFR part 46. Approval has been given under expedited review category 7 as outlined below:

(7) *Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.*

Your application will expire on March 28, 2013. If your research will continue beyond this date, complete the relevant portions of the IRB Renewal Application. If you wish to modify the application, complete the Modification of an Approved Protocol Form. Changes in this study cannot be initiated without IRB approval, except when necessary to eliminate apparent immediate hazards to participants. When the study closes, complete the appropriate portions of the IRB Request for Study Closure Form.

Should you need to submit any further correspondence regarding this proposal, please include the above application number.

Good luck with your research.

Sincerely,

Carpantato T. Myles, MSM, CIM
Director & Research Compliance Officer
Office of Research Compliance
The University of Alabama



**UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS**

I. Identifying information

Names:	Principal Investigator Jeffrey Carver	Second Investigator Debra McCallum	Third Investigator Rosanna E. Guadagno
Department:	Computer Science	SSRC	Psychology
College:	Engineering	Arts & Sciences	Arts & Science
University:	Alabama	Alabama	Alabama
Address:	Box 870290	391A Nott Hall	Box 870348
Telephone:	8-9829	8-3820	8-7803
FAX:	8-0219		8-8648
E-mail:	carver@cs.ua.edu	dmccallu@as.ua.edu	rosanna@ua.edu

Title of Research Project: Survey of Open Source Developers

Date Submitted: March 23, 2012

Funding Source: N/A

Type of Proposal	<input type="checkbox"/> New	<input type="checkbox"/> Revision	<input checked="" type="checkbox"/> Renewal Please attach a renewal application	<input type="checkbox"/> Completed	<input type="checkbox"/> Exempt
Please attach a continuing review of studies form					
Please enter the original IRB # at the top of the page					

UA faculty or staff member signature: _____

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):Type of Review: Full board X Expedited**IRB Action:**

- Rejected Date: _____
 Tabled Pending Revisions Date: _____
 Approved Pending Revisions Date: _____

Approved-this proposal complies with University and federal regulations for the protection of human subjects.

Approval is effective until the following date: 3/28/2013

- Items approved: Research protocol (dated _____)
 Informed consent (dated _____)
 Recruitment materials (dated _____)
 Other (dated _____)

Approval signature _____

Date 3/29/2012

Office for Research

Institutional Review Board for the
Protection of Human Subjects



October 17, 2012

Jeffrey Carver, Ph.D.
Department of Computer Science
College of Engineering
Box 870290

Re: IRB #: EX-12-CM-067, "Survey of Code Review Practices in Free \ Open
Source Software Projects"

Dear Dr. Carver:

The University of Alabama Institutional Review Board has granted approval for
your proposed research.

Your application has been given exempt approval according to 45 CFR part
46.101(b)(2) as outlined below:

*(2) Research involving the use of educational tests (cognitive, diagnostic,
aptitude, achievement), survey procedures, interview procedures or observation
of public behavior, unless (i) information obtained is recorded in such a manner
that human subjects can be identified, directly or through identifiers linked to the
subjects; and (ii) any disclosure of the human subjects' responses outside the
research could reasonably place the subjects at risk of criminal or civil liability
or be damaging to the subjects' financial standing, employability, or reputation.*

This approval expires on October 16, 2013. If the study continues beyond that
date, you must complete the appropriate portion of the Continuing Review Form.
If you modify the application, please complete the Modification of an Approved
Protocol Form. Changes in this study cannot be initiated without IRB approval,
except when necessary to eliminate apparent immediate hazards to participants.
When the study closes, please complete the appropriate Closure form.

Should you need to submit any further correspondence regarding this application,
please include the assigned IRB application number.

Good luck with your research.

Sincerely,

[Redacted]
Carpantato T. Myles, MSM, CIM
Director & Research Compliance Officer
Office of Research Compliance
The University of Alabama



358 Rose Administration Building
Box 870127
Tuscaloosa, Alabama 35487-0127
(205) 348-8461
FAX (205) 348-7189
TOLL FREE (877) 820-3066

IRB Project #:

**UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS**

I. Identifying information

Names:	Principal Investigator Jeffrey Carver	Second Investigator Amiangshu Bosu	Third Investigator
Department:	Computer Science	Computer Science	
College:	Engineering	Engineering	
University:	Alabama	Alabama	
Address:	Box 870290	2025 Shelby Hall	
Telephone:	8-9829	8-1528	
FAX:	8-0219		
E-mail:	carver@cs.ua.edu	asbosu@crimson.ua.edu	

Title of Research Project: Survey of Code Review Practices in Free \ Open Source Software projects

Date Submitted: October 02, 2012

Funding Source: N/A

Type of Proposal	<input checked="" type="checkbox"/> New	<input type="checkbox"/> Revision	<input type="checkbox"/> Renewal Please attach a renewal application	<input type="checkbox"/> Completed	<input type="checkbox"/> Exempt
Please attach a continuing review of studies form					
Please enter the original IRB # at the top of the page					

UA faculty or staff member signature: [REDACTED]

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):

Type of Review: _____ Full board Expedited

IRB Action:

- Rejected Date: _____
 Tabled Pending Revisions Date: _____
 Approved Pending Revisions Date: _____

Approved-this proposal complies with University and federal regulations for the protection of human subjects.

Approval is effective until the following date: 10/16/13

Items approved: Research protocol (dated 10/17/12)
 Informed consent (dated 10/17/12)
 Recruitment materials (dated 10/17/12)

Approval signature [REDACTED] Date _____

**UNIVERSITY OF ALABAMA
Informed Consent for a Research Study**

Dr. Jeffrey Carver, and Amiangshu Bosu from the University of Alabama, is conducting a study called "Survey of Code Review Practices in Free/Open-source Software Communities". They wish to understand how peer code review practices in Free / Open Source software communities help the project in various ways.

Taking part in this study involves completing a web survey that will take about 20 minutes. This survey contains questions about your project and your experience as a code review participant.

All data gathered will be anonymous from the time of collection. No identifying information will be associated with the data. There is no risk associated with lack of privacy. Only Dr. Carver and his graduate students will have access to the data. The data will be stored on Dr. Carver's password protected computer and secured survey hosting site (SurveyMonkey). Only summarized data will be presented at meetings or in publications.

There will be no direct benefits to you for participating in the survey. However, the findings will provide the researchers important insights into peer code review practices in Free/Open-source communities.

There are no risks to participating in this study. You may skip any questions you do not want to answer.

If you have questions about this study, please contact Dr. Carver at (205)-348-9829 or carver@cs.ua.edu. If you have questions about your rights as a research participant, contact Ms. Tanta Myles (the University Compliance Officer) at (205) 348-8461 or toll-free at 1-877-820-3066. If you have complaints or concerns about this study, file them through the UA IRB outreach website at http://osp.ua.edu/site/PRCO_Welcome.html. Also, if you participate, you are encouraged to complete the short Survey for Research Participants online at this website. This helps UA improve its protection of human research participants.

YOUR PARTICIPATION IS COMPLETELY VOLUNTARY. You are free not to participate or stop participating any time before you submit your answers.

If you understand the statements above, are at least 19 years old, and freely consent to be in this study, click on the _____ (CONTINUE or I AGREE) button to begin.

UNIVERSITY OF ALABAMA IRB
CONSENT FORM APPROVED: 10/17/12
EXPIRATION DATE: 10/16/13

Email Solicitation:

Subject: Research Invitation: Survey of Code Review practices in Free \ Open Source Software projects

Dr. Jeffrey Carver, and Amiangshu Bosu, University of Alabama are conducting a survey on Code Review practices in Free \ Open Source software projects. This survey seeks to understand peer code review practices and their benefits on Free \ Open Source software projects. You must be at least 19 years of age to complete the survey. The survey should take approximately 20 minutes to complete.

Here is the link to the survey: <https://www.surveymonkey.com/s/code-review-survey>

This survey has been approved by The University of Alabama IRB board.

UA IRB Approved Document
Approval date: 10/17/12
Expiration date: 10/16/13

Office for Research

Institutional Review Board for the
Protection of Human Subjects



August 2, 2013

Amiangshu Bosu
Department of Computer Science
College of Engineering
Box 870290

Re: IRB # EX-13-CM-080: "Survey of Code Review Practices in Commercial Projects"

Dear Mr. Bosu,

The University of Alabama Institutional Review Board has granted approval for your proposed research.

Your application has been given exempt approval according to 45 CFR part 46.101(b)(2) as outlined below:

(2) Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless:

- i. information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and*
- ii. any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.*

This approval expires on August 1, 2014. If the study continues beyond that date, you must complete the IRB Renewal Application. If you modify the application, please complete the Modification of an Approved Protocol form. Changes in this study cannot be initiated without IRB approval, except when necessary to eliminate apparent immediate hazards to participants. When the study closes, please complete the Request for Study Closure form.

Please reproduce the IRB-approved consent language for the online survey.

Should you need to submit any further correspondence regarding this application, please include the assigned IRB application number.

Good luck with your research.

Sincerely,



Carpantato T. Myles, MSM, CIM
Director & Research Compliance Officer
Office for Research Compliance
The University of Alabama

IRB Project #:

EX-13-CM-080

UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS

I. Identifying information

	Principal Investigator	Second Investigator	Third Investigator
Names:	Amiangshu Bosu	Dr. Jeffrey C. Carver	
Department:	Computer Science	Computer Science	
College:	Engineering	Engineering	
University:	Alabama	Alabama	
Address:	Box 870290	Box 870290	
Telephone:	8-1528	8-9829	
FAX:		8-0219	
E-mail:	asbosu@crimson.ua.edu	carver@cs.ua.edu	

Title of Research Project: Survey of Code Review Practices in Commercial Projects

Date Submitted: June 18, 2013

Funding Source: N/A

Type of Proposal	<input checked="" type="checkbox"/> New	<input type="checkbox"/> Revision	<input type="checkbox"/> Renewal Please attach a renewal application	<input type="checkbox"/> Completed	<input type="checkbox"/> Exempt
Please attach a continuing review of studies form					
Please enter the original IRB # at the top of the page					

UA faculty or staff member signature: [REDACTED] _____

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):

Type of Review: _____ Full board _____ Expedited

IRB Action:

- Rejected Date: _____
 Tabled Pending Revisions Date: _____
 Approved Pending Revisions Date: _____
 Approved-this proposal complies with University and federal regulations for the protection of human subjects.

Approval is effective until the following date: 8/1/2014

- Items approved: Research protocol (dated _____)
 Informed consent (dated _____)
 Recruitment materials (dated _____)
 Other (dated _____)

Approval signature [REDACTED] Date 8/2/2013

**UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS**

I. Identifying information

	Principal Investigator	Second Investigator	Third Investigator
Names:	Amiangshu Bosu	Jeffrey Carver	
Department:	Computer Science	Computer Science	
College:	Engineering	Engineering	
University:	Alabama	Alabama	
Address:	Box 870290	Box 870290	
Telephone:	8-1528	8-9829	
FAX:	8-0219	8-0219	
E-mail:	asbosu@crimson.ua.edu	carver@cs.ua.edu	

Title of Research Project: Survey of Code Review Practices in Commercial Projects

Date Submitted: July 11, 2014

Funding Source: National Science Foundation

Type of Proposal	<input type="checkbox"/> New	<input type="checkbox"/> Revision	<input checked="" type="checkbox"/> Renewal Please attach a renewal application
		<input type="checkbox"/> Completed	<input type="checkbox"/> Exempt
Please attach a continuing review of studies form			
Please enter the original IRB # at the top of the page			

UA faculty or staff member signature: [REDACTED]

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):

Type of Review: _____ Full board _____ Expedited

IRB Action:

- Rejected Date: _____
- Tabled Pending Revisions Date: _____
- Approved Pending Revisions Date: _____

Approved-this proposal complies with University and federal regulations for the protection of human subjects.

Approval is effective until the following date:

- Items approved: Research protocol (dated _____)
 Informed consent (dated _____)
 Recruitment materials (dated _____)

Approval signature [REDACTED]

Date

*7/16/2014**[Signature]*

Informed Consent for a Research Study

Dr. Christian Bird from Microsoft Research, together with Dr. Jeffrey Carver, and Amiangshu Bosu from the University of Alabama, are conducting a study called "Survey of Code Review Practices in Commercial projects". They wish to understand how peer code review practices in Microsoft help the projects in various ways.

Taking part in this study involves completing a web survey that will take about 10 minutes. This survey contains questions about your project and your experience as a code review participant.

All data gathered will be anonymized after collection. No identifying information will be associated with the data. There is no risk associated with lack of privacy. Dr. Bird will anonymize and filter the survey data according to Microsoft policies before sharing with collaborators at the University of Alabama. Only summarized data will be presented at meetings or in publications.

There will be no direct benefits to you for participating in the survey. However, the findings will provide the researchers important insights into peer code review practices in closed-source projects.

There are no risks to participating in this study. You may skip any questions you do not want to answer.

This study has been approved the University of Alabama Institutional Review Board for the Protection of Human Subjects. If you have questions about this study, please contact Dr. Bird at --- or Dr. Carver at (205) -348-9829 (carver@cs.ua.edu) . If you have questions about your rights as a research participant, contact Ms. Tanta Myles (Research Compliance Officer at the University of Alabama) at (205) 348-8461 or toll-free at 1-877-820-3066. If you have complaints or concerns about this study, file them through the UA IRB outreach website at http://osp.ua.edu/site/PRCO_Welcome.html. Also, if you participate, you are encouraged to complete the short Survey for Research Participants online at this website. This helps UA improve its protection of human research participants.

YOUR PARTICIPATION IS COMPLETELY VOLUNTARY. You are free not to participate or stop participating any time before you submit your answers.

If you understand the statements above, are at least 19 years old, and freely consent to be in this study, click on the _____ (CONTINUE or I AGREE) button to begin.

UNIVERSITY OF ALABAMA IRB
CONSENT FORM APPROVED: 8/2/2013
EXPIRATION DATE: 5/1/2014

Office for Research

Institutional Review Board for the
Protection of Human Subjects



July 16, 2014

Amiangshu S. Bosu, M.S.
Department of Computer Science
College of Engineering
The University of Alabama
Box 870290

Re: IRB # EX-12-CM-080-R1 "Survey of Code Review Practices in Commercial Projects"

Dear Mr. Bosu:

The University of Alabama Institutional Review Board has granted approval for your renewal application.

Your renewal application has been given exempt approval according to 45 CFR part 46.101(b)(2) as outlined below:

(2) Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless:

(i) information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

Your application will expire on July 15, 2015. If your research will continue beyond this date, complete the relevant portions of Continuing Review and Closure From. If you wish to modify the application, complete the Modification of an Approved Protocol Form. When the study closes, complete the appropriate portions of FORM: Continuing Review and Closure.

Should you need to submit any further correspondence regarding this proposal, please include the assigned IRB application number.

Good luck with your research.

Sincerely,



Carpanato T. Myles, MSM, CIP
Director & Research Compliance Officer
Office for Research Compliance
The University of Alabama

358 Rose Administration Building
Box 870127
Tuscaloosa, Alabama 35487-0127
(205) 348-8461
FAX (205) 348-7189
TOLL FREE (877) 820-3066