Modeling Modern Code Review Practices in Open Source Software Development Organizations

Amiangshu Bosu
Department of Computer Science
University of Alabama
Tuscaloosa, AL, USA
asbosu@ua.edu

ABSTRACT

Many Open Source Software (OSS) communities has adopted peer code review as an effective quality improvement practice. The informal, regular, and tool-based code review process has been called 'modern code review'. There has not been much research on the modern code review process. This dissertation aims to model the this code review process adopted by the software organizations through 1)understanding, 2) improving, and 3) providing suggestions. I have planned seven steps to achieve the research goal, which includes literature review, exploring OSS code review practices, mining OSS code review repositories, comparing code review metrics across projects, surveying OSS code review participants, analyzing the effectiveness of peer code reviews to prevent security vulnerabilities, and social network analysis.

So far, I have completed first three steps and made significant progress towards the next three steps. The research has contributed an empirical evidence that code review helps building accurate peer impressions between the code review participants. The planned future contributions include providing effective strategies to perform peer code review, evaluating the effectiveness of peer code review to prevent security vulnerabilities, and influence of project characteristics /social network structure on the peer code review process.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*productivity, programming teams*

General Terms

Measurement, Design, Human Factors, Verification

Keywords

peer code review, peer impression, open source,inspection

1. INTRODUCTION

Software inspections, in which developers subject their code to review by peers or other stakeholders to identify defects, was developed by Michael Fagan in 1976 [12]. Since then, it has been established as an effective quality improvement practice [32, 9]. 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 [18]. Conversely, many mature, successful Open Source Software (OSS) projects have recently adopted peer code reviews (informal software inspection) as an important quality assurance mechanism. We define, **Peer Code Review** as, *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*.

There are multiple ways to perform a code review, but they all have a similar goal of evaluating certain properties of newly written or revised code. Many of the OSS project coding guidelines state that all the code changes must be approved through peer code review before merging into the main project branch. Some of the well-known OSS projects, that require mandatory code review include: Apache, Chromium browser, Chromium OS, Mozilla, QT, and Android. Even many commercial organizations are also adopting the peer code review practice [2]. However, there are marked differences between Fagan-style code inspection and peer code-review practices adopted by the software organizations. First, peer code-review are practiced asynchronously using mailing list or code-review tools. Bacchelli and Bird called this practice as *Modern Code Review*, and identified three key characteristics of this code review process: 1) informal, 2) tool-based, and 3) occurs regularly in practice [2].

Despite the recent widespread adoption of the peer code review process in both OSS and commercial projects, there has not been much research on modern code review. Therefore, the following three topics warrant further research.

1. **Understand**
   - the code review practices adopted in different types of organizations
   - the common types of flaws identified during code review
   - both technical and non-technical benefits/drawbacks of code review
2. **Improve**
   - the process by suggesting solutions to commonly identified code review problems
   - the efficiency of code reviews by suggesting which types of code need more attention
3. **Suggest**
   - how different types of organizations can be benefited by adopting code review practice
   - the best strategies to review code changes
   - how to select reviewer for code changes

IDoESE '13 Baltimore, Maryland USA

In my dissertation, I plan to address research questions from each of the three areas to model the modern code review process. The remainder of the paper is organized as following. § 2 summarizes a motivating study. § 3 summarizes prior research results. § 4 describes the high level research questions, potential study designs, and studies already performed for my dissertation. Finally, § 5 mentions the areas where I would like to have feedback and concludes the paper.

2. PRELIMINARY MOTIVATING STUDY

The broader research goal of the current research project includes modeling the peer *impression formation* between the members of OSS communities. Peer impression is "the judgments that a person, called the perceiver, makes about another person, called the target". The formation of peer impressions primarily depends upon how well the perceiver is acquainted with the target and upon the personality traits of those two individuals [19].

Generally, OSS participants 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) [15, 17]. Due to the marked differences between FTF and CMC interactions [4], the impression formation process between OSS participants, who primarily collaborate using CMC, is different than the impression formation process between developers working in FTF settings. CMC participants often have difficulties forming accurate impression about the abilities of their peers. Yet, OSS participants consider those impressions about their peers very important during interaction[5]. Because peer impression is a major motivator for OSS participants and has great influence on the interaction between the OSS peers, it is important to understand the peer impression formation process within the OSS communities. However, there is a lack of knowledge about that process [21].

To better understand peer impression formation and evolution within 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 the study, we primarily focused on five forms of peer impressions: 1) productivity, 2) competency, 3) easy or difficult to work with, 4) expertise, and 5) trustworthiness. To gather data to investigate these issues, we designed an online survey that focused on communication, collaboration, and peer evaluation in OSS communities. We sent the survey to the developer mailing lists of 48 popular, successful OSS projects (i.e., projects with more than 50,000 downloads) and received a total of 115 responses.

The full details of the survey is under review, but can be viewed in a technical report [5]. First, we focused on the impressions of the OSS participants about working with their peers. Most of the participants (approximately 61%) indicated having positive experiences with their peers on the FLOSS projects. The rest of the participants indicated negative experiences due to poor management, difficulty in collaboration, delayed communication, hard to get involved, and time-zone difference. We also found that volunteers were significantly more likely to report negative experiences about working with peers than were the paid participants.

Second, we focused on understanding whether impressions of peer developers had any effect on peer interactions. The respondents indicated that their impression of a peer's level of expertise and trustworthiness were significant factors in their interactions with that peer. Third, about half (55%) of the respondents believed that their peers had an accurate impression of their ability, 31% were not sure, but thought that peers had an accurate impression of their ability and 13% thought that peers did not have an accurate impression of their ability. Some of the reasons given for the responses included the fact that because the development process and communication is done in the open, people should be able to form accurate impressions. Conversely, for those that were not sure other's had an accurate view, they stated that because there was not much time spent together, as might occur on a co-located project, peers may have incorrect impressions.

Fourth, we wanted to determine which factors OSS participants consider most important to judge the productivity or competency of their peers. There were three types of factors: Work Style (i.e., creativity, accuracy, response time and efficiency), Historical Factors (i.e., handle, background and online profile) and Project Contribution (i.e., quality and quantity of committed code and comments, critical fixes made, important design decisions made, written communication and other work products). The respondents overall rated the Project Contribution factors as being the most important and the Historical Factors as being the least important. Therefore, the results show that for judging the competency of a peer, developers prefer to use information about that person's actual contributions to the project rather than other interpersonal factors. Our detailed analysis of the results revealed that when a participant performs a significant task (e.g. developing a critical module) with accuracy and other participants find that module easily understandable for integration or collaboration, those participants form a positive impression of the original participant.

Finally, we asked respondents to rate the importance of various factors in determining if someone was easy or difficult to work with. Similar to the other results, we found the Coding-related factors (i.e. poor quality code, buggy features, codes violating design integrity) were the most important factors for identifying difficult to work with peers.

The results of this survey provide some important insights into the current beliefs of OSS developers from a wide range of projects. These results suggest that coding-related factors influence the peer impression formation most in OSS projects. Therefore, we believe those types of interactions where participants can have the opportunity to judge a peer's code or creativity should be crucial in peer impression formation. Since, code review facilitates direct interactions to judge a peer's code, it should therefore serve to best support peer impression formation. This results are my primary motivations to model the peer code review process in OSS communities.

3. RELATED WORK

After observing time loss and difficulties in scheduling formal code inspection meetings, Votta raised the question whether formal meetings are really needed for code inspections [31]. He suggested that the reviewer and the author can correspond using verbal, written, or electronic medium without actually meeting physically. Later, OSS communities championed the informal code review process using electronic communication (i.e. mailing list).

Rigby and German are the first among the researchers to examine the informal code review process in OSS communities [27]. After exploring code review policies and process of 11 OSS projects, they generalized the code review process practiced in those communities. 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). To characterize the code review practice in Apache project, they calculated 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). Comparing similar code review metrics from five OSS projects, Asundi and Jayant concluded that the code review processes and characteristics vary across different OSS projects due to the differences in age and culture [1]. In a subsequent study, Rigby et al. analyzed the two types of peer code review practiced in the Apache. The found that the CTR is faster

than the RTC but is not significantly less efficient in bug detection. They conclude that the secret behind the efficient and effective peer review techniques of the Apache is the "early, frequent reviews of small, independent, complete contributions conducted asynchronously by a potentially large, but actually small, group of self-selected experts" [28]. Later, Rigby and Storey studied the code review decision mechanisms in five OSS projects using broadcast based review(i.e. sending patches to a mailing list for review). They conclude 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 those OSS projects are aware that too many opinions on a patch leads to ionated and unproductive discussions, therefore they try to avoid that to ensure efficient broadcast based code review [29]. Finally, based on their study results from OSS code review process, Rigby et al. suggested how software companies can adapt the peer code review process effectively [26]. The works of Rigby et al. [27, 28, 29, 26] lays an important foundation towards building a theory of OSS code review process.

The success of the OSS code review process has encouraged many of the commercial projects to adopt peer code review practices, which is evident by some of the recent publications [30, 26, 2, 3]. Contrary to OSS projects, code review participants at Microsoft use both synchronous and asynchronous communication mediums for code review and they found code review dialogs essential for understand code changes and design rational later. They expressed a need to retain code review communications for later information needs [30]. After surveying code review participants at Microsoft about the expectations, motivations, actual outcomes, and challenges of modern code review, Bacchelli and Bird found that the motivations of the developers for code review do not match exactly with the top outcomes, as the top motivations were finding defects, code improvement, alternative solutions, and knowledge transfer but code improvements, understanding, and social communication were among the top outcomes. Knowledge transfer was mentioned but was much lower on the outcome list. The respondents mentioned understanding the code changes as the major challenge for doing a code review. Based on their analysis, they suggest that although major motivations for code review is finding defects, the major benefits of code review is beyond that [2]. To reduce the time required for code review, Balachandran introduced a tool named Review Bot which is able to automate the checks for coding standard violations and common defect patterns. The tools can also suggest potential reviewers based on a learning algorithm that he developed[3].

Despite very limited research on peer code review in prior years, there has been a number of publications [2, 3, 7, 25] on peer code review within the last one year indicating growing interests among the software engineering researchers about peer code review.

4. RESEARCH GOAL AND DESIGN

This dissertation aims to 1) better understand the OSS peer code review process, and 2) based on those understandings identify best practices, and 3) suggest improvements. To achieve those goals, I have defined three high level research questions. I plan to explore the answers for these questions in my dissertation.

- **HRQ1: How do participants perform peer code review?**
  - *RQ1.1: How efficient are the peer code review practices in different OSS projects?*
  - *RQ1.2: What are the social network structures of different OSS developer communities based on peer code review interactions?*

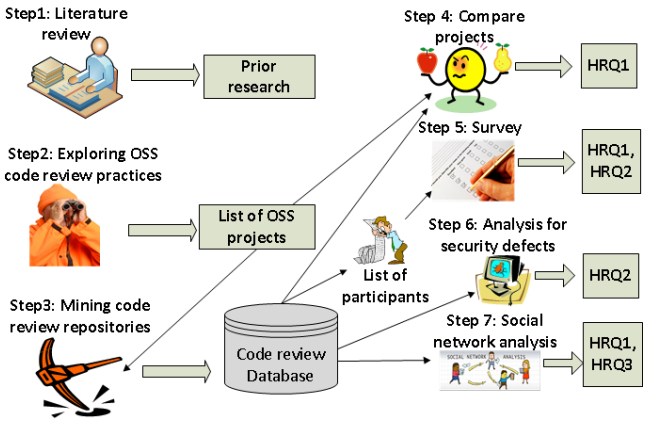

Figure 1: Research Design

- **HRQ2: How do OSS projects benefit from peer code review?**
  - *RQ2.1: How do OSS participants benefit from participating in peer code review?*
  - *RQ2.2: How does peer code review influence impression formations between code review participants?*
  - *RQ2.3: How effective is peer code review for preventing different types of security vulnerabilities?*
- **HRQ3: How do project/participant characteristics influence the peer code review process?**

I have designed seven steps to explore those research questions. I have already made significant progress on the first three steps. I am currently working on the next three steps. Following subsections provide a summary of the seven steps. Figure 1 illustrates the seven steps which are detailed in the following subsections. Table 1 summarizes the planned studies to provide a timeline.

4.1 Reviewing Literature

This step involves reviewing literature related to peer code review. There has not been much prior research work on peer code review. Section 3 summarizes the prior research. However, Code inspections have some similarity with peer code review and has been studied over a long period. Therefore, I am also looking into prior research on code inspections to determine which results from code inspection research may be also applicable to peer code review. That will also help me to formulate detailed research questions.

4.2 Exploring OSS Code Review Practices

This step involves exploring the code review practices of different OSS communities. I have made significant progress on this task. Following paragraphs describes the process I followed in this step.

First, I made a list of OSS communities those practice peer code review. The initial list contained 44 OSS communities.

Second, I explored the code review policies of different OSS communities in my list. OSS projects have varying policies regarding code review. Many OSS communities list their code review policies on project website(i.e. Mozilla policy [1], MediaWiki[2]). For example, three Apache core members should approve a code change before merging into the trunk [27]. Many of the projects require authors submitting every code change for peer review, while

[1] https://wiki.mozilla.org/Firefox/Code_Review
[2] https://www.mediawiki.org/wiki/Gerrit/Code_review

Table 1: Dissertation plan

Study Name	Research Method	Research Question(s)	Planned Publication(s)	Status	Timeline
1. Comparing Code Review Practices (§4.4)	Data mining, case study	HRQ1	2	Data collected, One publication [6]	May 2014
2. Surveying Code Review Participants (§4.5)	Data mining, Online survey	HRQ1, HRQ2	2	Survey conducted, One publication [7]	December 2013
3. Analyzing the Effectiveness of Peer Code Review to Prevent Security Vulnerabilities (§4.6)	Data mining, case study	HRQ2	1	Study conducted, In submission	September 2013
4. Social Network Analysis of Code Review Interactions (§4.7)	Data mining, case study	HRQ1, HRQ3	1	Data collection planned	September 2014

others require only critical or new member's code changes to be reviewed.

Third, I explored how different projects perform code reviews. Previously, most of the OSS projects practiced mailing list based code reviews. Linux kernel[3] still uses mailing list based reviews. Mozilla has integrated code review into bug management system (BugZilla). However, recent trends are more towards using specialized web-based tools to manage code review requests.

Finally, I listed the code review tools used by different OSS projects. Most of the OSS code review repositories are publicly accessible. After exploring the code review repositories, I listed the repository type,access URL, and the level of code review activities for each of the OSS projects from the initial list. I prepared the second list of projects, after excluding 18 projects from the initial list as those 18 projects had a small number of (less than 1000) code review requests posted in the repository.

4.3 Mining Code Review Repositories

This step involves mining publicly accessible data from the OSS code review repositories. Most of the popular OSS projects have started using tools to manage code review requests. Three code review tools are popular among the OSS communities. Table 2 presents a brief summary of those three tools. To better analyze the code review practices, I have written three Java applications (i.e. Gerrit-Miner, RietVeld-Miner, and ReviewBoard-Miner) to mine the code review repositories managed by the three tools. Those applications are able to mine all the publicly accessible data for each of the review requests posted in the code review repositories managed by the three tools and populate MySQL databases.

Among the three code review tools, only RietVeld provides publicly accessible API, which returns data in JSON (JavaScript Object Notation) format. The RietVeld-Miner application parses the JSON data returned by the RietVeld API to get details information about a code review request. Gerrit does not provide a documented API. However, after analyzing the Gerrit managed web pages, I reverse-engineered some JSON-RPC (JSON Remote Procedure Call) calls between web browsers and Gerrit web-servers. The Gerrit-Miner application simulates similar JSON-RPC calls to mine Gerrit repositories. The ReviewBoard-Miner application downloads HTML pages for each review requests and parses the code review request data using XPATH analysis.

Using the three miner applications, I have mined code review repositories hosted by 26 OSS communities. Table 3 lists the projects, types of the repository management systems, number of review requests mined, and total number of code review participants in those projects. Please note that, these numbers are according to data mined from the repositories during the last week of May 2013.

4.4 Comparing Code Review Practices

[3] https://www.kernel.org/doc/Documentation/SubmittingPatches

In this step, I defined a number of metrics to measure the efficiency of code review process. Table 4 describes those metrics. We conducted a small scale study [6] using the data mined from Asterisk, and MusicBrainz server code-review repositories. The purpose of the study was to determine the applicability of our code review metrics to compare OSS code review practices. Fist, we found that code authors were not submitting all code changes for peer review. More than 80% of the review requests were responded by two or less reviewers. The top committers in the projects were also the top code review contributors. Most of the review requests received prompt feedback within a day; however, some requests waited much longer for feedback. These two projects had very high code approval ratio. Code authors modified the code changes according to the code review feedback and received approval for merging in the trunk.

Table 3: Code Review Repositories Mined

Project	Code Review Management Tool	Total No. of Code Review Requests	Total No. of Code Review Participants
Android	Gerrit	18,110	1,642
Android ARM	Gerrit	1,393	30
Asterisk	ReviewBoard	1689	155
Chromium Browser	RietVeld	270,410	4,571
Chromium OS	Gerrit	47,392	1050
Coreboot	Gerrit	2,955	94
Couchbase	Gerrit	24,640	147
Cyanogenmod	Gerrit	31,226	1,655
Eclipse	Gerrit	11,363	454
Gerrit	Gerrit	4,207	271
Gromacs	Gerrit	1,578	44
KDE	ReviewBoard	17,712	768
Kitware	Gerrit	10,549	548
LibreOffice	Gerrit	3198	207
Linaro	Gerrit	3,151	75
MediaWiki	Gerrit	59,738	412
Membase	Gerrit	24,640	147
MusicBrainz	ReviewBoard	2177	34
OmapZoom	Gerrit	31,458	1,032
OpenAFS	Gerrit	9,369	87
OpenStack	Gerrit	26,766	1,154
OVirt	Gerrit	13,647	224
QT project	Gerrit	53,303	956
ReviewBoard	ReviewBoard	1611	337
Scilab	Gerrit	11,127	50
Typo3	Gerrit	18,673	511

Table 2: Popular Open Source Code Review Tools

Tool Name	Project Homepage	License	Latest Version	Written in
Gerrit	https://code.google.com/p/gerrit/	Apache License v2	2.6.1	Java, Servlet, GWT
ReviewBoard	http://www.reviewboard.org/	MIT License	1.7.11	Python, Django
RietVeld	https://code.google.com/p/rietveld/	Apache License v2	-	Python, Django

Table 4: Code Review Efficiency Metrics

Metrics	Definition
Reviewers per request	How many code reviewers respond to a code review request?
Comments per request	How many comments are posted on each code review request?
First feedback interval	Time elapsed between posting a code review request and its first review feedback
Review interval	Time elapsed between posting a code review request and its approval
Approval ratio	Percentage of code changes approved for inclusion in the trunk

The observed differences between the code review metrics between the two projects works as motivation to conduct a large scale study to compare the code review practices of different OSS communities. The code review data mined from 26 OSS communities (Table 3) will help conduct this proposed study. The primary goal of this study is to determine the correlations between the code-review metrics and different project characteristics (e.g. project size, project age, number of contributors, technology, and sponsorship).

In the preliminary study, we also noticed some code review requests were getting prompted feedback, while other review requests had to wait for a longer period of time. I hypothesize that the reputations (i.e. core contributors) and prior interactions (i.e. reviewed prior code changes) between the author and the reviewer impacts the prompt/delayed feedback. Studying this hypothesis is the secondary goal of the proposed study. This study will help me to answer the first high level research question (*RQ1.1*). For the study, I will select the projects to ensure diversity between project ages, domains, languages, and sizes to combat the possible external threats to validity,

4.5 Surveying Code Review Participants

For the first two high level research questions (*HRQ1*, and *HRQ2*), opinions of the code review participants can provide crucial insights. From the mined code review data (Table 3), I prepared a list of OSS participants who are actively participating peer code review. We designed a web-based survey[4] focusing on the first three high level research questions. We sent out the survey to persons from our list during February - March 2013. We received encouraging responses from 287 code review participants. Partial results of that survey will appear in the proceedings of ESEM -2013 [7]. The objectives of the survey was to understand: 1) the OSS code review process, 2) expectations of the code review process, and 3) how code review impacts peer impression formation.

Since, the survey focused on the behavioral aspects of code review participants, we followed well-regarded social and behavioral research methods to design our survey [13, 10]. We used established scales from Psychology, Information science, and Organizational Behavior to create the survey questions. We primarily focused on better understanding how code review helps four aspects of peer impression formation: trust, reliability, perception of expertise, and friendship. The results indicate that there is indeed a high level of trust, reliability, perception of expertise, and friendship between OSS peers who have participated in code review for a period of time. The results of the survey provides an empirical evidence to support the anecdotal evidence that code review helps building accurate peer impressions between the code review participants (*RQ2.2*). Because code review involves examining someone else's code, unsurprisingly, peer code review helped most in building a perception of expertise between code review partners. Based on this result, we suggest that DSD teams suffering from a lack cohesion can adopt code reviews, not only to improve software quality, but also to improve the impressions formation among teammates.

A number of commercial organizations are also practicing peer code review [2] (e.g. Microsoft, Facebook, VMWare, and Google). Many of the commercial software development companies worldwide have adopted distributed software development (DSD), which is largely similar to OSS development [16]. Therefore, we expect many study results obtained from OSS projects might be applicable to closed-source DSD projects. The results of the code review survey encouraged us to replicate it in a closed source environment. We have identified a organization that agreed to replicate our survey. We plan to send this survey to code review participants from both co-located and distributed projects inside that organization. This replication will help us to compare and contrast between the code review practices in OSS, closed-source DSD, and co-located projects.

There were several open-ended questions in the survey about code review process and expectations from code review. We are analyzing those open-ended responses using a systematic qualitative data analysis process. Two REU (i.e. Research Experience for Undergraduate, sponsored by National Science Foundation) students examined the responses to extract the general themes associated with each response. We discussed these themes to develop a coding scheme for each question. Using these coding schemes, the REU students are working independently to code the responses. After they complete the individual coding, they will compare the results to identify any discrepancies. The coders will then sit together to resolve those discrepancies through discussion. For the unresolved discrepancies, I plan to meet with the two coders and provide a tie-breaking vote after listening to the rationale for each of the inconsistent codes. The results of this analysis will help to gain more insights into the first two high level research questions (*HRQ1*, and *HRQ2*).

To remove potential threats to validity, we carefully designed our survey. Multiple reviewers from different disciplines reviewed the survey questions. We conducted three pilot studies to ensure high quality survey questions. We enforced several reliability and validity measures to ensure higher confidence in the survey results. Finally, to eliminate potential coding bias, we are using two coders to analyze the open-ended responses.

4.6 Analyzing the Effectiveness of Peer Code Review to Prevent Security Vulnerabilities

Security vulnerabilities are specific source code flaws that allow attackers to expose, alter, disrupt, or destroy sensitive information [11]. Even though there are numerous examples of exploited

[4] http://asbosu.students.cs.ua.edu/data/code-review/code-review-survey.pdf

security flaws that resulted in significant financial losses or other serious problems, users still report numerous security vulnerabilities to publicly available vulnerability databases (e.g. Common Vulnerabilities and Exposures - CVE).

Peer code review has been suggested as an important practice for detecting and correcting security bugs [22]. For example, expert reviewers can identify code that contains potential security vulnerabilities and help the author eliminate the security flaws or abandon the vulnerable code. Moreover, peer code review can identify attempts to insert malicious code into the codebase. According to McGraw, the longer it takes to detect and fix a security vulnerability the higher the overall cost associated with that vulnerability [23]. Therefore, peer code review can reduce the cost of creating secure software by helping developers eliminate security defects earlier when it is less expensive.

During our analysis of the peer code review process in OSS communities [6], we observed that code reviewers often provide detailed comments and suggestions about potential bugs. Therefore, we can find indications of security flaws by text-mining the code review comments. For example, if a reviewer suspects the presence of a potential buffer overflow, his/her review comments will most probably contain either *buffer*, *overflow* or both. The results of our preliminary study [8] verified the feasibility of this approach. Using text-mining, we have built and validated a list of keywords (Table 5) to mine review comments relevant to different types of security defects. Following steps describe the approach I followed to build the set of empirically validated keywords. I used the R - text mining (tm) package for the analysis.

1. Create a initial set of keywords relevant to security defects.
2. Mine the code review repositories and populate database.
3. Search the database using initial set of keywords and build a CSV file (Corpus). Each entry in the csv file is a code review comments containing at least one of the predetermined keywords.
4. Because, many of the comments and texts contains code snippet, apply identifier splitting rules on the corpus. (i.e. isBufferFull becomes "is Buffer Full" or read_string becomes "read string").
5. Clean the corpus. Remove whitespace, punctuation, and numbers. Convert all words to lowercase. Create list of tokens for each document (i.e. row in csv) in the Corpus.
6. Apply porter stemming algorithm to find the stem of each of the tokens. (i.e. buffer, buffered, buffering all becomes buffer).
7. Create a Document Term matrix from the corpus.
8. Determine the words those co-occurred frequently with each of our predetermined keywords.
9. Manually inspect all the frequently co-occurring words, to determine which keywords should be added to the predetermined keywords list. The last row of Table 5 lists the keywords, we added after the test-mining.

We queried the mined code-review databases (Table 3) to create a list of code-review comments containing at least one of the listed keywords. Then, two REU students inspected independently to reject any comments clearly unrelated to security defects. We discarded a review comment at this step only if both students considered it unrelated to security defects. Each of the remaining review comments that passed the first inspection phase, indicates a potential discussion about security vulnerabilities in the corresponding code review request. Using the remaining review comments, we have created a list of review requests with potential discussion about security vulnerabilities. We are now manually inspecting each review request from that list. We are inspecting both the code review discussions and associated patches to determine whether any security vulnerabilities were prevented during the code review process. We are collecting a set of metrics (e.g. author, reviewer, date, and vulnerability type, author experience, reviewer experience, project) for the code review requests having evidences of preventing potential security vulnerabilities. Analyzing those metrics, we expect to answer following research questions.

Table 5: Keywords Associated with Security Flaws

Vulnerability Type	Keywords
Buffer overflow	buffer, overflow, stack
Format string	format, string, printf, scanf
Integer overflow	integer, overflow, signedness, widthness, underflow
Cross site scripting	cross site, CSS, XSS
SQL injection	SQL, SQLI, injection
Command injection	command, injection
Improper access control	improper, unauthenticated, gain access, permission
Denial of Service/Race	denial service, DOS, race
Cross site request forgery	cross site, request forgery, CSRF, XSRF, forged
Common	security, vulnerability, vulnerable, hole, exploit, attack, bypass, crash
Common (added later)	threat, expose, breach, violate, fatal, blacklist, overrun

1. How many vulnerabilities were prevented by the peer code-review process?
2. Which type(s) of vulnerabilities were detected by the peer code-review process?
3. What were the outcomes of the peer code-review suggestions (e.g. abandoned or modified) for the code changes those had vulnerabilities?
4. Which types of reviewers (in terms of background or expertise) were most likely to detect vulnerabilities?
5. Which type of author (in terms of background or expertise) was most likely to introduce vulnerabilities?
6. Are the vulnerable code changes significantly differ (in terms of avg. size,and avg. complexity) from other code changes?
7. Is peer code review effective to secure a software project?

The answers to those questions will help to answer the fourth high level research question (*RQ2.3*). To address any potential reviewer bias, two students are working independently in each of the phases. The students will discuss among themselves to resolve any disagreements. I will work as the third reviewer to provide tie breaking votes on disagreements they are unable to resolve. Because, the results of this study depends using correct set of keywords, we also plan to validate the completeness of our keyword set. We will manually examine a set of randomly selected review requests those do not contain any of our keywords. Among this set, the existence of no or very few review requests with evidence of prevented potential security vulnerabilities using code review process will validate our keyword set.

Table 6: Social Network Analysis Metrics

Metric	Definition	Usage
Degree Centrality	Number of edges attached to a node	Determining popularity of a node in a network
Betweenness Centrality	Number of shortest paths that pass through a node divided by all shortest paths in the network	Identifying the critical nodes for communication channels
Closeness Centrality	Mean length of all shortest paths from a node to all other nodes	Determining how quickly information flows within a network
Average Distance	The average of all shortest paths in a network	Determining how far at any two nodes will be on average
Density	Ratio of the number of edges in the network over the total number of possible edges between all pairs of nodes	Determining how well connected a network is
Clustering	A node's clustering coefficient is the density of its neighborhood	Determing which network topology the network structure fits
Core-periphery Fitness	Position of the high degree nodes	Determing whether high degree nodes tend to have stronger ties with other high degree nodes
Degree Distribution	Probability distribution of the degrees between all the nodes of a network	Understanding the network topology (e.g. random graph, poisson distribution, power law)

4.7 Social Network Analysis

A social network is a theoretical construct that represents the relationships between individuals, organization, or societies. These networks are typically modeled using a graph structure consisting of vertices and edges. Vertices typically represent individuals or organizations. An edge connecting two vertices represents some type of relationships between the two individuals or organizations. Social network analysis focuses on studying social network graphs to understand the patterns of interactions and the relative positions of individuals in a social settings [14]. Social network analysis is being widely used by researchers from a broad range of fields: sociology, biology, antrhopology, communication studies, information science, organizational studies, and psychology. Several researchers have also used social network analysis of sociotechnical interaction (i.e. mailing list, code repository, and bug repository) graphs to model the social structure of OSS communities.

In OSS communities, community members communicate primarily via various project mailing-lists. Most OSS projects have multiple mailing lists that each serve a different purpose (e.g. user support, bug report, announcements, and development). Among those lists the developers use the *development* mailing-list to discuss project designs, development directions, and other pending issues (both technical and non-technical). Because all developers subscribe to the development mailing list, all members of the community are involved in the mailing list threads (i.e. reads the message and can participant). As a result, it would not be accurate to conclude that two participants interact with each other simply because they are active in the same mailing-list thread. To accurately measure the level of interaction between two participants, we have to consider other types of sociotechnical interactions between those participants (e.g. interactions in the bug repository, code review, or interactions in the code repository). Each type of interaction provides a different view into the social structure of OSS projects, therefore we anticipate that the social networks generated from these interactions will differ.

During our analysis for the study to compare the code review practices of OSS communities [6], we found evidences of strong social network ties between the code review participants. Social network graphs indicated a few central persons being responsible for most of the interactions in the network. Again, we found that in some cases certain participants were more likely to review code written by other participants. Furthermore, in most cases review requests between participants who were already acquainted had a quicker turn around than requests between participants who were not acquainted. These results suggested that a study of the social network created by code review interactions could provide fruitful insights into OSS projects.

I will use the data mined from code review repositories (Table 3) to build the social network graphs. In these graphs, individual participants will be represented by the nodes. The weight of an edge between two nodes in the graph will be calculated based upon the number of code review requests between two participants. If two participants have not interacted through code review, there will not be an edge between the nodes representing those developers. For each project, we will also build other types of social network graphs that have been used in previous research, i.e. based upon bug repository information and upon code repository information. To build the bug repository social network I will to use a similar approach as Long et al.[20] where nodes represent developers and edge weights represent the number of mutual bug requests. I will build the code repository network using a similar approach as Meneely et al.[24] where nodes represent developers and edge wights represent the number of mutual code commit on same file/module.

For the social networks, I will calculate widely used social network analysis (SNA) measures (Table 6) to compare the sociotechnical networks generated based upon code review information, code repository information and bug repository information (RQ1.2). As the interaction between participants varies based upon the type of interaction measured, I expect to observe some interesting differences among the three graphs. For example, developers who work on the same module have better ideas on the appropriateness of new changes to those module and are more likely to review each other's code. Therefore, I expect that code review social network may be similar to code repository social network. However, the bug interaction social networks may differ because there are patch submitters who only fix bugs and do not have the privileges to make new changes. Furthermore, each review request typically has at least one of the most experienced developers of a module as a reviewer. Therefore the experienced developers will likely be more prevalent in the code review interactions. I expect that the code review social networks will be more centralized around a small number of developers than the other two types of social networks. The results of this analysis will help to understand the OSS community social structure and the interactions between the OSS participants better.

Finally, I will use to correlate the SNA metrics with the code review metrics (Table 4) to find out the impact of social network structure on the efficiency of code review process (HRQ3).

5. AREAS OF CONCERN & CONCLUSION

This proposed dissertation work aims to model the modern peer

code review process. The planned contributions of this dissertation work include providing empirical evidences regarding the technical/ non-technical benefits of peer code review, empirically validate the effectiveness of peer code review to prevent security vulnerabilities, determine the best code review practices, and comparative analysis of the OSS social network structures based on code review interaction. This work will also help to understand the peer impression formation process between OSS participants, which is considered difficult as well as different from co-located projects.

There are several issues of which the author of this dissertation would like to get the most advice on. First, for the four proposed study designs, what are the other possible threats to validity and how the author can modify the study designs to combat those? Second, code inspections is similar to peer code review in some respects and has been studied over the years. The author would like to know if any prior research on code inspections can be useful to build a model of peer code review. Finally, apart from the proposed research questions, are there any other interesting research questions that the author should consider for his dissertation.

6. ACKNOWLEDGMENTS

This research is partially supported by the NC State Science of Security lablet, and the National Science Foundation under grant numbers REU-1156563, and VOSS-1322276.

7. REFERENCES

[1] J. Asundi and R. Jayant. Patch review processes in open source software development communities: A comparative case study. In *40th Annual Hawaii Intl. Conf. on System Sciences, 2007. HICSS 2007.*, pages 166c–166c. IEEE, 2007.

[2] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proc. of the 2013 Intl. Conf. on Soft. Eng.*, pages 712–721. IEEE Press, 2013.

[3] V. Balachandran. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proc. of the 2013 Intl. Conf. on Soft. Eng.*, pages 931–940. IEEE Press, 2013.

[4] J. A. Bargh and K. Y. A. McKenna. The internet and social life. *Annual Review of Psychology*, 55(1):573–590, 2004.

[5] A. Bosu, J. Carver, L. Hochstein, and R. Guadagno. Peer impressions in open source organizations: A survey. Technical Report SERG-2011-3, Department of Computer Science, University of Alabama, 2011.

[6] A. Bosu and J. C. Carver. Peer code review in open source communities using reviewboard. In *Proc. of the 4th ACM Wksp. on Evaluation and Usability of Prog. Lang. and Tools*, pages 17–24, New York, NY, USA, 2012. ACM.

[7] A. Bosu and J. C. Carver. Impact of peer code review on peer impression formation: A survey. In *Proc. of the 7th ACM/IEEE Intl. Symposium on Empirical Soft. Engineering and Measurement*, page To Appear. IEEE, 2013.

[8] A. Bosu and J. C. Carver. Peer code review to prevent security vulnerabilities: An empirical evaluation. In *Soft. Security and Reliability (SERE), 2013 IEEE Seventh Intl. Conf. on*, page to Appear, 2013.

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

[10] R. F. DeVellis. *Scale development: Theory and applications*, volume 26. Sage, 2011.

[11] M. Dowd, J. McDonald, and J. Schuh. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.

[12] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Sys. Journal*, 15:182–211, 1976.

[13] A. Fink. *The survey handbook*, volume 1. Sage, 2003.

[14] L. Freeman. *The development of social network analysis: A study in the sociology of science*, volume 1. Empirical Press Vancouver, 2004.

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

[16] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Soft. Eng.*, pages 188–198. IEEE Computer Society, 2007.

[17] S. L. Jarvenpaa and D. E. Leidner. Communication and trust in global virtual teams. *Journal of Computer-Mediated Communication*, 3(4):791–815, 1998.

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

[19] D. A. Kenny. *Interpersonal perception: a social relations analysis*. Guilford Press, 2004.

[20] Y. Long and K. Siau. Social network structures in open source software development teams. *Journal of Database Management (JDM)*, 18(2):25–40, 2007.

[21] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Proc. of the 2013 Conf. on Computer supported cooperative work*, CSCW '13, pages 117–128, New York, NY, USA, 2013. ACM.

[22] G. McGraw. Soft. security. *IEEE Security Privacy*, 2(2):80 – 83, Mar-Apr 2004.

[23] G. McGraw. Automated code review tools for security. *Computer*, 41(12):108 –111, Dec. 2008.

[24] A. Meneely and L. Williams. Socio-technical developer networks: should we trust our measurements? In *33rd Intl. Conf. on Soft. Eng. (ICSE), 2011*, pages 281 –290, may 2011.

[25] M. Mukadam, C. Bird, and P. C. Rigby. Gerrit software code review data from android. In *Proc. of the Tenth Intl. Workshop on Mining Soft. Repositories*, pages 45–48. IEEE Press, 2013.

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

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

[28] P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: a case study of the apache server. In *Proc. of the 30th Intl. Conf. on Soft. Eng.*, pages 541–550. ACM, 2008.

[29] P. C. Rigby and M.-A. Storey. Understanding broadcast based peer review on open source software projects. In *Proc. of the 33rd Intl. Conf. on Soft. Eng.*, ICSE '11, pages 541–550, New York, NY, USA, 2011. ACM.

[30] A. Sutherland and G. Venolia. Can peer code reviews be exploited for later information needs? In *Soft. Eng.-Companion Volume, 2009. ICSE-Companion 2009. 31st Intl. Conf. on*, pages 259–262. IEEE, 2009.

[31] L. G. Votta Jr. Does every inspection need a meeting? In *ACM SIGSOFT Soft. Eng. Notes*, volume 18, pages 107–114. ACM, 1993.

[32] K. E. Wiegers. *Peer reviews in Software: A practical guide*. Addison-Wesley Boston, 2002.