Imed Hammouda, Björn Lundell, Greg Madey and Megan Squire (Eds.)

Proceedings of the Doctoral Consortium at the 13th International Conference on Open Source Systems

Buenos Aires, Argentina, 22 May 2017

UNIVERSITY OF SKÖVDE

Skövde University Studies in Informatics 2017:1

Hammouda, I., Lundell, B., Madey, G. and Squire, M. (Eds.) Proceedings of the Doctoral Consortium at the 13th International Conference on Open Source Systems, Skövde University Studies in Informatics 2017:1, ISSN 1653-2325, ISBN 978-91-983667-1-6, University of Skövde, Skövde, Sweden.

Skövde University Studies in Informatics 2017:1
ISSN 1653-2325
ISBN 978-91-983667-1-6

www.his.se

UNIVERSITY
OF SKÖVDE

Proceedings of the Doctoral Consortium at the 13th International Conference on Open Source Systems, 2017

Edited by:

Imed Hammouda
Chalmers and University of Gothenburg, Sweden

Björn Lundell
University of Skövde, Sweden

Greg Madey
University of Notre Dame, USA

Megan Squire
Elon University, USA

Preface

The last two decades have witnessed a tremendous growth in the interest and diffusion of Free/Libre and Open Source Software (FLOSS) technologies, which has transformed the way organizations and individuals create, acquire and distribute software and software-based services. The Open Source Systems conference as its premier publication venue has reached its thirteenth edition this year.

To facilitate new researchers with an arena to present and receive feedback on their research, the Open Source Systems conference has had a Doctoral Consortium for several years. The principle objective of the consortium is to provide doctoral students the opportunity to present their research at various stages of production – from early drafts of their research design to near completion of their dissertation – in a forum where they can receive constructive feedback from a community of interested scholars and other students as they work to finish their degree. This volume contains the six papers, each of which was reviewed by members of the program committee. After the reviews, authors were given the opportunity to revise their papers based on the input they received from the reviewers and participants who provided feedback during the event.

This volume contains the revised versions of the papers, which were presented and discussed at the Doctoral Consortium at the Thirteenth International Conference on Open Source Systems, in Buenos Aires, Argentina in May 2017.

We wish to thank the reviewers and members of the Program Committee of the Doctoral Consortium who have provided valuable feedback on the papers. We also thank all Ph.D. students and senior researchers for their participation. Finally, we are grateful for the financial support (award number 1639136) provided by the U.S. National Science Foundation (NSF).

Imed Hammouda
Björn Lundell
Greg Madey
Megan Squire

Program Committee

Kevin Crowston	Syracuse University, USA
Imed Hammouda	Chalmers & University of Gothenburg, Sweden
Juho Lindman	Chalmers & University of Gothenburg, Sweden
Björn Lundell	University of Skövde, Sweden
Greg Madey	University of Notre Dame, USA
Megan Squire	Elon University, USA

Table of Contents

Longitudinal Statistical Analysis of Open Source Software Development Forks

Amirhosein "Emerson" Azarbakht

Oregon State University
School of Electrical Engineering & Computer Science
4099 Kelley Engineering Center, Corvallis OR 97331, USA
azarbaam@oregonstate.edu
http://eecs.oregonstate.edu/people/azarbakht

Abstract. *Social interactions are a ubiquitous part of our lives, and the creation of online social communities has been a natural extension of this phenomena. Free and Open Source Software (FOSS) development efforts are prime examples of how communities can be leveraged in software development, where groups are formed around communities of interest, and depend on continued interest and involvement.*

Forking in FOSS, either as an non-friendly split or a friendly divide, affects the community. Such effects have been studied, shedding light on how forking happens. However, most existing research on forking is post-hoc. In this study, we focus on the seldom-studied run-up to forking events. We propose using statistical modeling of longitudinal social collaboration graphs of software developers to study the evolution and social dynamics of FOSS communities. We aim to identify measures for influence and the shift of influence, measures associated with unhealthy group dynamics, for example a simmering conflict, in addition to early indicators of major events in the lifespan of a community.

We use an actor-oriented approach to statistically model the changes a FOSS community goes through in the run-up to a fork. The model represents the tie formation, breakage, and maintenance. It uses several (more than two, up to 10) snapshots of the network as observed data to estimate the influence of several statistical effects on formation of the observed networks. Exact calculation of the model is not trivial, so, instead we simulate the changes and estimate the model using a Markov Chain Monte Carlo approach.

When we find a well-fitting model, we can test our hypothesis about model parameters, the contributing effects using T-tests and Multivariate Analysis of Variance Between Multiple Groups (Multivariate ANOVA). Our method enables us to make meaningful statements about whether the network dynamics depends on particular parameters/effects with a p-value, indicating the statistical significance level.

This approach may help predict formation of unhealthy dynamics, which is the first step toward a model that gives the community a heads-up when they can still take action to ensure the sustainability of the project.

1 Introduction

Social networks are a ubiquitous part of our social lives, and the creation of online social communities has been a natural extension of this phenomena. Social media plays an important role in software engineering, as software developers use them to communicate, learn, collaborate and coordinate with others [56]. Free and Open Source Software (FOSS) development efforts are prime examples of how community can be leveraged in software development, where groups are formed around communities of interest, and depend on continued interest and involvement to stay alive [39].

Community splits in free and open source software development are referred to as forks, and are relatively common. Robles et al. [47] define forking as “when a part of a development community (or a third party not related to the project) starts a completely independent line of development based on the source code basis of the project.”

Although the bulk of collaboration and communication in FOSS communities occurs online and is publicly accessible for researchers, there are still many open questions about the social dynamics in FOSS communities. Projects may go through a metamorphosis when faced with an influx of new developers or the involvement of an outside organization. Conflicts between developers’ divergent visions about the future of the project may lead to forking of the project and dilution of the community. Forking, either as an acrimonious split when there is a conflict, or as a friendly divide when new features are experimentally added, affect the community [10].

Previous research on forking ranges from the study by Robles et al. [47] that identified 220 significant FOSS projects that have forked over the past 30 years, and compiled a comprehensive list of the dates and reasons for forking (listed in Table 1, and depicted by frequency in Figure 1), to the study by Baishakhi et al. [8] on post-forking porting of new features or bug fixes from peer projects. It encompasses works of Nyman on developers’ opinions about forking [41], developers motivations for performing forks [36], the necessity of code forking as tool for sustainability [40], and Syeed’s work on sociotechnical dependencies in the BSD projects family [57].

Most existing research on forking, however, is post-hoc. It looks at the forking events in retrospect and tries to find the outcome of the fork; what happened after the fork happened; what was the cause of forking, and such. The run-up to the forking events are seldom studied. This leaves several questions unanswered: Was it a long-term trend? Was the community polarized, before forking happened? Was there a shift of influence? Did the center of gravity of the community change? What was the tipping point? Was it predictable? Is it ever predictable? We are missing that context.

Additionally, studies of FOSS communities tend to suffer from an important limitation. They treat community as a static structure rather than a dynamic process. Longitudinal studies on open source forking are rare. To better understand and measure the evolution, social dynamics of forked FOSS projects,

and integral components to understanding their evolution and direction, we need new and better tools. Before making such new tools, we need to gain a better understanding of the context. With this knowledge and these tools, we could help projects reflect on their actions, and help community leaders make informed decisions about possible changes or interventions. It will also help potential sponsors make informed decisions when investing in a project, and throughout their involvement to ensure a sustainable engagement.

Identification is the first step to rectify an undesirable dynamic before the damage is done. A community that does not manage growing pains may end up stagnating or dissolving. Managing growing pains is especially important in the case of FOSS projects, where near half the project contributors are volunteers [21]. Identification of recipes for success or stagnation, sustainability or fragmentation may lead to a set of best practices and pitfalls.

I propose to use temporal social network analysis to study the evolution and social dynamics of FOSS communities. Specifically, we propose using a longitudinal exponential family random graph statistical model to investigate the driving forces in formation and dissolution of communities. Additionally, to complement the statistical study, we propose doing a qualitative interview study for validating the findings. With these techniques we aim to identify better measures for influence, shifts of influence, measures associated with unhealthy group dynamics, for example a simmering conflict, in addition to early indicators of major events in the lifespan of a community. One set of dynamics we are especially interested in, are those that lead FOSS projects to fork.

Table 1: The main reasons for forking as classified by Robles and Gonzalez-Barahona [47]

Reason for forking	**Example forks**
Technical (Addition of functionality)	Amarok & Clementine Player
More community-driven development	Asterisk & Callweaver
Differences among developer team	Kamailio & OpenSIPS
Discontinuation of the original project	Apache web server
Commercial strategy forks	LibreOffice & OpenOffice.org
Experimental	GCC & EGCS
Legal issues	X.Org & XFree

2 Related Work

The free and open source software development communities have been studied extensively. Researchers have studied the social structure and dynamics of team communications [11][23][27][28][35], identifying knowledge brokers and associated activities [53], project sustainability [35][40], forking [39], requirement

Table 2: The frequency of main reasons for forking as classified by Robles and Gonzalez-Barahona [47]

Reason	**Frequency**
Technical	60 (27.3%)
Discontinuation of the original project	44 (20.0%)
More community-driven development	29 (13.2%)
Legal issues	24 (10.9%)
Commercial strategy forks	20 (9.1%)
Differences among developer team	16 (7.3%)
Experimental	5 (2.3%)
Not Found	22 (10.0%)

satisfation [18], their topology [11], their demographic diversity [31], gender differences in the process of joining them [30], and the role of age and the core team in their communities [2][3][**?**][7][17][59]. Most of these studies have tended to look at community as a static structure rather than a dynamic process [16]. This makes it hard to determine cause and effect, or the exact impact of social changes.

Post-forking porting of new features or bug fixes from peer projects happens among forked projects [8]. A case study of the BSD family (i.e., FreeBSD, OpenBSD, and NetBSD, which evolved from the same code base) found that 10-15% of lines in BSD release patches consist of ported edits, and on average 26-58% of active developers take part in porting per release. Additionally, They found that over 50% of ported changes propagate to other projects within three releases [8]. This shows the amount of redundant work developers need to do to synchronize and keep up with development in parallel projects.

Visual exploration of the collaboration networks in FOSS communities was the focus of a study that aimed to observe how key events in the mobile-device industry affected the WebKit collaboration network over its lifetime. [58] They found that *coopetition* (both competition and collaboration) exists in the open source community; moreover, they observed that the “firms that played a more central role in the WebKit project such as Google, Apple and Samsung were by 2013 the leaders of the mobile-devices industry. Whereas more peripheral firms such as RIM and Nokia lost market-share” [58].

The study of communities has grown in popularity in part thanks to advances in social network analysis. From the earliest works by Zachary [60] to the more recent works of Leskovec et al. [32][33], there is a growing body of quantitative research on online communities. The earliest works on communities was done with a focus on information diffusion in a community [60]. The study by Zachary investigated the fission of a community; the process of communities splitting into two or more parts. They found that fission could be predicted by applying the Ford-Fulkerson min-cut algorithm [20] on the group’s communication graph; “the unequal flow of sentiments across the ties” and dis-

criminatory sharing of information lead to subcommunities with more internal stability than the community as a whole.[60]

The dynamic behavior of a network and identifying key events was the aim of a study by Asur et al [1]. They studied three DBLP co-authorship networks and defined the evolution of these networks as following one of these paths: a) Continue, b) k-Merge, c) k-Split, d) Form, or e) Dissolve. They defined four possible transformation events for individual members: 1) Appear, 2) Disappear, 3) Join, and 4) Leave. They compared groups extracted from consecutive snapshots, based on the size and overlap of every pair of groups. Then, they labeled groups with events, and used these identified events [1].

Table 3: The behavioral measures used by Asur et al. [1]

Metrics	**Meaning**
Stability	Tendency of a node to have interactions with the same nodes over time
Sociability	Tendency of a node to have different interactions
Influence	Number of followers a node has on a network and how its actions are copied and/or followed by other nodes. (e.g., when it joins/leaves a conversation, many other nodes join/leave the conversation, too)
Popularity	Number of nodes in a cluster (how crowded a sub-community is)

The communication patterns of free and open source software developers in a bug repository were examined by Howison et al. [27]. They calculated out-degree centrality as their metric. Out-degree centrality measures the proportion of times a node contacted other nodes (outgoing) over how many times it was contacted by other nodes (incoming). They calculated this centrality over time "in 90-day windows, moving the window forward 30 days at a time." They found that "while change at the center of FOSS projects is relatively uncommon," participation across the community is highly skewed, following a power-law distribution, where many participants appear for a short period of time, and a very small number of participants are at the center for long periods. Our proposed approach is similar to theirs in how we form collaboration graphs. Our approach is different in terms of our project selection criteria, the metrics we examine, and our research questions.

The tension between diversity and homogeneity in a community was studied by Kunegis et al. [31]. They defined five network statistics, listed in Table 4, used to examine the evolution of large-scale networks over time. They found that except for the diameter, all other measures of diversity shrunk as the networks matured over their lifespan. Kunegis et al. [31] argued that one possible reason could be that the community structure consolidates as projects mature.

Table 4: The measures of diversity used by Kunegis et al. [31]

Network property	**Network is diverse when**	**Diversity Measures**
Paths between nodes	Paths are long	Effective diameter
Degrees of nodes	Degrees are equal	Gini coefficient of the degree distribution
Communities	Communities have similar sizes	Fractional rank of the adjacency matrix
Random walks	Random walks have high probability of return	Weighted spectral distribution
Control of nodes	Nodes are hard to control	Number of driver nodes

Community dynamics was the focus of a more recent study by Hannemann and Klamma [24] on three open source bioinformatics communities. They measured ”age” of users, as starting from their first activity and found survival rates and two indicators for significant changes in the core of the community. They identified a survival rate pattern of 20-40-90%, meaning that only 20% of the newcomers survived after their first year, 40% of the survivors survived through the second year, and 90% of the remaining ones, survived over the next years. As for the change in the core, they suggested that a falling maximum betweenness in combination with an increasing network diameter as an indicator for a significant change in the core, e.g., retirement of a central person in the community. Our initial network-specific study built on their findings, and the evolution of betweenness centralities and network diameters for the projects in our study are explained in the following sections.

3 Research Goals

Social interactions reflect the changes the community goes through, and so, it can be used to describe the context surrounding a forking event. Social interactions in FOSS can happen, for example, in the form of mailing list email correspondence, bug report issue follow-ups, and source code co-authoring.

We consider the following three of the seven main reasons for forking [47] to be socially related: (1) Personal differences among developer team, (2) The need for more community-driven development, and (3) Technical differences for addition of functionality.

By socially-related, we mean, the forking categories that should have left traces in the developers’ interactions data. Such traces may be identified using longitudinal modeling of the interactions, without digging into the contents of the communications. These three reasons are (1) Personal differences among developer team, (2) The need for more community-driven development, and (3) Technical differences for addition of functionality.

Table 5: The socially-related reasons for forking

Reason	**Frequency**
Differences among developer team	16 (7.3%)
More community-driven development	29 (13.2%)
Technical	60 (27.3%)

As an example of how these traces of forking can be identified, if a fork occurred because of a desire for "more community-driven development", we should see interaction patterns in the collaboration data showing a strongly-connected core that is hard to penetrate for the rest of the community (i.e. the power stayed in the hands of the same people throughout, as developers joined and left.)

In this study, we plan to analyze, quantify and visualize how the community is structured, how it evolves, and the degree to which community involvement changes over time.

Specifically, our overall research objective is to identify these traces/social patterns associated with different types of undesirable forking?

In the following and in section 3, we will discuss our research objectives and research questions in depth.

Do forks leave traces in the collaboration artifacts of open source projects in the period leading up to the fork?
To study the properties of possible social patterns, we need to verify their existence. More specifically, we need to check whether the possible social patterns are manifested in the the collaboration artifacts of open source projects, e.g., mailing list data, issue tracking systems data, source code data. This is going to be accomplished by statistical modeling of developer interactions as explained in more detail in section 4.

Do different types of forks leave different types of traces?
If forks leave traces in the collaboration artifacts, do forks exhibit different social patterns? Are there patterns that exemplify these categories? For example, is there a prototypical "personal differences" fork collaboration pattern? If so, do different forking reasons have distinctly different social patterns associated with them? Is a project labeled as a "technical differences" fork only a "technical differences" fork? Or, alternatively, can they be a mix of several reason categories?

We are going to investigate this by statistical modeling of the interaction graphs, as illustrated in Figure 1.

What are the key indicators that let us distinguish between different types of forks?
What quantitative measure(s) can be used as an early warning sign of an inflection point (fork)? Are there metrics that can be used to monitor the odds

of change, (e.g. forking-related patterns), ahead of time? This will be accomplished by statistical modeling of developer interactions as explained in more detail in section 4.

To validate what our quantitative approach finds, and to account and check for possible confounding factors, we will interview and survey people from the studied forked projects. We will also analyze the sentiments in the content of the messages send and received by the top contributors of the project in the month leading to the forking events will be analyzed.

4 Methodology

Figure 1 shows the overview of our methodology.

Detecting change patterns, requires gathering relevant data, cleaning it, and analyzing it. In the following subsections, we describe the proposed process in detail.

4.1 Data Collection

4.1.1 Data Sources The data sources to collect are **a)** developer mailing lists, where developers' interact by sending and receiving emails, and **b)** Source-code repository contribution logs, where developers interact by modifying the code. The sociograms were formed based on interactions among developers in any of the preceding data sources.

For the purpose of our study, we gathered data for 13 projects, in three categories of forking, plus a control group. The time period for which data was collected is one year leading to when the decision to break-up (fork) happened. This should capture the social context of the run-up to the forking event.

4.1.2 Data Cleaning and Wrangling Mailing list data was cleaned such that the sender and receiver email ID case-sensitivity differences would be taken into account. The Source Code repository version control logs were used to capture the source code activity levels of the developers who had contributed more than a few commits. The set of the developers who had both mailing list activity and source code repository activity formed the basis of the socio-grams we used in our analysis.

4.2 Sociogram Formation for Statistical Modeling

Social connections and non-connections can be represented as graphs, in which the nodes represent actors (developers) and the edges represent the interaction(s) between actors or lack thereof. Such graphs can be a snapshot of a network – a static sociogram – or a changing network, also called a dynamic

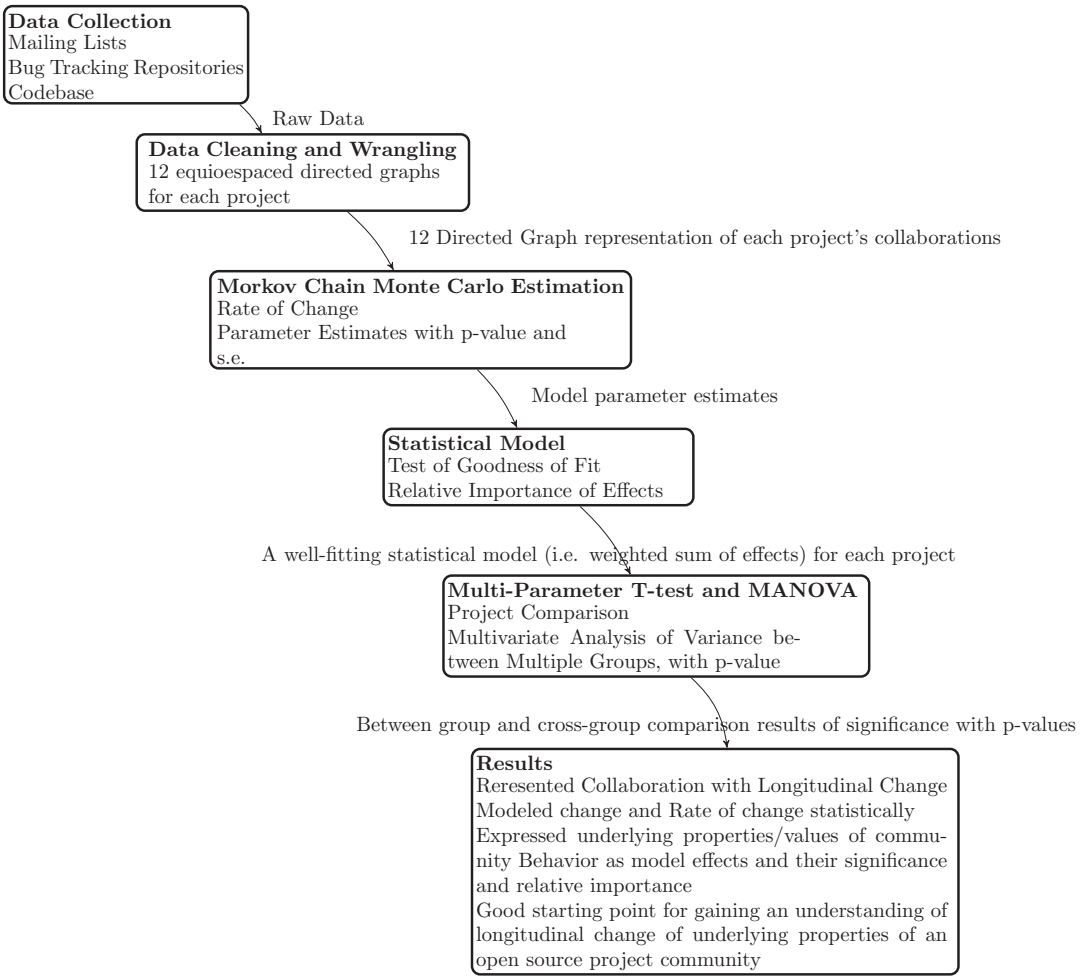


Fig. 1: The methodology in a glance

sociogram. In this phase, we process interactions data to form a communication sociogram of the community.

Two types of analysis can be done on sociograms: Either a *cross-sectional* study, in which only one snapshot of the network is looked at and analyzed; or a *longitudinal* study, in which several consecutive snapshots of the network are looked at and studied. We are interested in patterns in the run-up to forks, therefore, unlike most existing research on forking, we did a longitudinal study.

We formed 10 equispaced consecutive time-window snapshots of the sociograms for the community, using the mailing list interaction data and the source

code repository commit activity data. These socio-grams were used to find a well-fitting statistical model that would explain how they changed from time-window t_1 through time-window t_{10}.

4.3 Validation

4.3.1 Qualitative Study: Interviews and Survey To validate what our quantitative approach finds, and to account and check for possible confounding factors, we need to compare it to what people remember of the situation. This validation check requires interviewing and surveying people from the studied forked projects. Semi-structured interviews need to be conducted, with as many developers from the forked projects, till the interviewers reach a point of saturation (i.e., when no new information is gained by doing more interviews), as possible. These semi-structured interviews will be recorded, transcribed, and coded according to the statistical model's covariates, to find overlapping and common patterns.

4.3.2 Sentiment Analysis To complement the study, the content of the messages send and received by the top contributors of the project in the month leading to the forking events will be analyzed. This data will be used as one of the developers' individual attributes in our statistical modeling.

4.3.3 Cross-Validation To test and validate our quantitative findings, we will model projects with "unknown" (or treated as "unknown") forking history using the same longitudinal modeling method.

The new model can then be compared to the "known" models in each forking category, using the ANOVA test. This comparison can provide new insights as to which category of forking reasons is the likely reason for forking or not-forking of the "unknown" projects. In this way, we may extrapolate about new projects' collaboration patterns.

References

1. Asur, S., S. Parthasarathy, and D. Ucar, (2009), "*An event-based framework for characterizing the evolutionary behavior of interaction graphs,*" ACM Trans. Knowledge Discovery Data. 3, 4, Article 16, (November 2009), 36 pages. 2009.
2. Azarbakht, A. and C. Jensen, "*Drawing the Big Picture: Temporal Visualization of Dynamic Collaboration Graphs of OSS Software Forks,*" Proc. 10th Int'l. Conf. Open Source Systems, 2014.
3. Azarbakht, A. and C. Jensen, "*Temporal Visualization of Dynamic Collaboration Graphs of OSS Software Forks,*" Proc. Int'l. Network for Social Network Analysis (INSNA) Sunbelt XXXIV Conf., 2014.
4. Azarbakht, A., "*Drawing the Big Picture: Analyzing FLOSS Collaboration with Temporal Social Network Analysis,*" Proc. 9th Int'l. Symp. Open Collaboration, ACM, 2013.

5. Azarbakht, A. and C. Jensen, "*Analyzing FOSS Collaboration & Social Dynamics with Temporal Social Networks,*" Proc. 9th Int'l. Conf. Open Source Systems Doct. Cons., 2013.
6. Azarbakht, A., "*Temporal Visualization of Collaborative Software Development in FOSS Forks,*" Proc. IEEE Symp. Visual Languages and Human-Centric Computing, 2014.
7. Azarbakht, E.A. and C. Jensen, "*Longitudinal Analysis of the Run-up to a Decision to Break-up (Fork) in a Community,*" Proc. 13th IFIP International Conference on Open Source Systems. Springer, Cham, 2017.
8. Baishakhi R., C. Wiley, and M. Kim, "*REPERTOIRE: a cross-system porting analysis tool for forked software projects,*" Proc. ACM SIGSOFT 20th Int'l. Symp. Foundations of Software Engineering, ACM, 2012.
9. Bastian, M., S. Heymann, and M. Jacomy, "*Gephi: an open source software for exploring and manipulating networks,*" Int'l AAAI Conf. on Weblogs and Social Media, 2009.
10. Bezrukova, K,, C. S. Spell, J. L. Perry, "*Violent Splits Or Healthy Divides? Coping With Injustice Through Faultlines,*" Personnel Psychology, Vol 63, Issue 3. 2010.
11. Bird, C., D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "*Latent social structure in open source projects,*" Proc. 16th ACM SIGSOFT Int'l. Symposium on Foundations of software engineering, ACM, 2008.
12. Brandes, U. "*A Faster Algorithm for Betweenness Centrality*", Journal of Mathematical Sociology 25(2):163-177, 2001.
13. Chakrabarti, D., and C. Faloutsos. "*Graph mining: Laws, generators, and algorithms,*" ACM Computing Surveys, 38, 1, Article 2, 2006.
14. Chen, C. and Liu, Lon-Mu, "*Joint Estimation of Model Parameters and Outlier Effects in Time Series,*" Journal of the American Statistical Association, 88, 284–297. 1993.
15. Coleman, J.S. "*Introduction to Mathematical Sociology,*" New York etc.: The Free Press of Glencoe. 1964.
16. Crowston, K., K. Wei, J. Howison, and A. Wiggins. "*Free/Libre open-source software development: What we know and what we do not know,*" ACM Computing Surveys, 44, 2, Article 7, 2012.
17. Davidson, J, R. Naik, A. Mannan, A. Azarbakht, C. Jensen, "*On older adults in free/open source software: reflections of contributors and community leaders,*" Proc. IEEE Symp. Visual Languages and Human-Centric Computing, 2014.
18. Ernst, N., S. Easterbrook, and J. Mylopoulos, "*Code forking in open-source software: a requirements perspective,*" arXiv preprint arXiv:1004.2889, 2010.
19. Feuerriegel S. and N. Proellochs. "*SentimentAnalysis: Dictionary-based sentiment analysis*", R package version 1.1-0. https://github.com/sfeuerriegel/SentimentAnalysis. 2016.
20. Ford, L. R. and D. R. Folkerson, "*A simple algorithm for finding maximal network flows and an application to the Hitchcock problem,*" Canadian Journal of Mathematics, vol. 9, pp. 210-218, 1957.
21. Forrest, D., C. Jensen, N. Mohan, and J. Davidson, "*Exploring the Role of Outside Organizations in Free/ Open Source Software Projects,*" Proc. 8th Int'l. Conf. Open Source Systems, 2012.
22. Fruchterman, T. M. J. and E. M. Reingold, "*Graph drawing by force-directed placement,*" Softw: Pract. Exper., vol. 21, no. 11, pp. 1129-1164, 1991.

23. Guzzi, A., A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. "*Communication in open source software development mailing lists,*" Proc. 10th Conf. on Mining Software Repositories, IEEE Press, 2013.
24. Hannemann, A and , R. Klamma "*Community Dynamics in Open Source Software Projects: Aging and Social Reshaping,*" Proc. Int. Conf. on Open Source Systems, 2013.
25. Heider, F. The Psychology of Interpersonal Relations. John Wiley & Sons. 1958.
26. Howison, J. and K. Crowston. "*The perils and pitfalls of mining SourceForge,*" Proc. Int'l. Workshop on Mining Software Repositories, 2004.
27. Howison, J., K. Inoue, and K. Crowston, "*Social dynamics of free and open source team communications,*" Proc. Int'l. Conf. Open Source Systems, 2006.
28. Howison, J., M. Conklin, and K. Crowston, "*FLOSSmole: A collaborative repository for FLOSS research data and analyses,*" Int'l. Journal of Information Technology and Web Engineering, 1(3), 17-26. 2006.
29. Krivitsky, P. N., and M. S. Handcock. "*A separable model for dynamic networks,*" Journal of the Royal Statistical Society: Series B (Statistical Methodology) 76, no. 1: 29-46. 2014.
30. Kuechler, V., C. Gilbertson, and C. Jensen, "*Gender Differences in Early Free and Open Source Software Joining Process,*" Open Source Systems: Long-Term Sustainability, 2012.
31. Kunegis, J., S. Sizov, F. Schwagereit, and D. Fay, "*Diversity dynamics in online networks,*" Proc. 23rd ACM Conf. on Hypertext and Social Media, 2012.
32. Leskovec, J., Kleinberg, J., and Faloutsos, C.: "*Graphs over time: densification laws, shrinking diameters and possible explanations,*" Proc. SIGKDD Int'l. Conf. Knowledge Discovery and data Mining, 2005.
33. Leskovec, J., K. J. Lang, A. Dasgupta, and M. W. Mahoney, "*Statistical properties of community structure in large social and information networks,*" Proc. 17th Int'l. Conf. World Wide Web, ACM, 2008.
34. Lopez-de-Lacalle, J. "*tsoutliers: Detection of Outliers in Time Series*", R package version 0.6-5. https://CRAN.R-project.org/package=tsoutliers, 2016.
35. Nakakoji, K., Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. "*Evolution patterns of open-source software systems and communities,*" Proc. Int'l. Workshop Principles of Software Evolution, ACM, 2002.
36. Mikkonen, T., L. Nyman, "*To Fork or Not to Fork: Fork Motivations in SourceForge Projects,*" Int'l. J. Open Source Softw. Process. 3, 3. July, 2011.
37. Noack, A., "*Energy models for graph clustering,*" J. Graph Algorithms Appl., vol. 11, no. 2, pp. 453-480, 2007.
38. Nowak, M. A. "*Five rules for the evolution of cooperation,*" Science 314, No. 5805: 1560-1563. 2006.
39. Nyman, L. , "*Understanding code forking in open source software,*" Proc. 7th Int'l. Conf. Open Source Systems Doct. Cons., 2011.
40. Nyman, L., T. Mikkonen, J. Lindman, and M. Fougère, "*Forking: the invisible hand of sustainability in open source software,*" Proc. SOS 2011: Towards Sustainable Open Source, 2011.
41. Nyman, L., "*Hackers on Forking,*" Proc. Int'l. Symp. on Open Collaboration, 2014.
42. Oh, W., Jeon, S., "*Membership Dynamics and Network Stability in the Open-Source Community: The Ising Perspective*" Proc. 25th Int'l. Conf. Information Systems. 2004.

43. Page, B, B. Sergey, R. Motwani and T. Winograd, "*The PageRank Citation Ranking: Bringing Order to the Web,*" Technical Report, Stanford InfoLab, 1999.
44. Proellochs, Feuerriegel and Neumann: "*Generating Domain-Specific Dictionaries Using Bayesian Learning*", Proceedings of the 23rd European Conference on Information Systems (ECIS 2015), Muenster, Germany, 2015.
45. R Core Team. "*R: A language and environment for statistical computing. R Foundation for Statistical Computing*", Vienna, Austria. URL https://www.R-project.org/. 2016.
46. Robins, G., P. Pattison, Y. Kalish, and D. Lusher. "*An introduction to exponential random graph (p*) models for social networks,*" Social networks 29, no. 2: 173-191. 2007.
47. Robles, G. and J. M. Gonzalez-Barahona, "*A comprehensive study of software forks: Dates, reasons and outcomes,*" Proc. 8th Int'l. Conf. Open Source Systems, 2012.
48. Rocchini, C. (Nov. 27 2012), Wikimedia Commons, Available: http://en.wikipedia.org/wiki/File:Centrality.svg, 2012.
49. Singer, L., F. Figueira Filho, B. Cleary, C. Treude, M. Storey, and K. Schneider. "*Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators,*" Proc. Conf. Computer supported cooperative work, ACM, 2013.
50. Snijders, T. AB. "*Markov chain Monte Carlo estimation of exponential random graph models,*" Journal of Social Structure 3, no. 2: 1-40. 2002.
51. Snijders, Tom AB. "*Models for longitudinal network data,*" Models and methods in social network analysis 1: 215-247. 2005.
52. Snijders, Tom AB., GG Van de Bunt, CEG Steglich, "*Introduction to stochastic actor-based models for network dynamics,*" Social networks 32 (1), 44-60. 2010.
53. Sowe, S., L. Stamelos, and L. Angelis, "*Identifying knowledge brokers that yield software engineering knowledge in OSS projects,*" Information and Software Technology, vol. 48, pp. 1025-1033, Nov 2006.
54. Spence, M. "*Job market signaling,*" Quarterly Journal of Economics, 87: 355-374. 1973.
55. Steglich, C., T. AB Snijders, and M. Pearson. "*Dynamic networks and behavior: Separating selection from influence,*" Sociological methodology 40, no. 1: 329-393. 2010.
56. Storey, M., L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "*The (R) Evolution of social media in software engineering,*" Proc. Future of Software Engineering, ACM, 2014.
57. Syeed, M. M., "*Socio-Technical Dependencies in Forked OSS Projects: Evidence from the BSD Family,*" Journal of Software 9.11 (2014): 2895-2909. 2014.
58. Teixeira, J., and T. Lin, "*Collaboration in the open-source arena: the webkit case,*" Proc. 52nd ACM conf. Computers and people research (SIGSIM-CPR '14). ACM, 2014.
59. Torres, M. R. M., S. L. Toral, M. Perales, and F. Barrero, "*Analysis of the Core Team Role in Open Source Communities,*" Int. Conf. on Complex, Intelligent and Software Intensive Systems, IEEE, 2011.
60. Zachary, W., "*An information flow model for conflict and fission in small groups,*" Journal of Anthropological Research, vol. 33, no. 4, pp. 452-473, 1977.

Efficient Bug Triage in Issue Tracking Systems

Anjali Goyal[1], Neetu Sardana[2]

Jaypee Institute of Information Technology, Noida, U.P., India.
[1]anjaligoyal19@yahoo.in, [2]neetu.sardana@jiit.ac.in

Abstract. Bug triaging is the process of designating a suitable developer for bug report who could make source code changes in order to fix the bug. Appropriate bug report assignment is important as it lowers the tossing path length and hence reduces the overall time and efforts involved in bug resolving. In this research work, our objective is to design a proficient recommendation framework for efficient bug triaging. In the literature, varied bug report assignment techniques exist. Research is still in progress to discover the most suitable bug report assignment technique. In this work, we first investigate the most appropriate bug triaging technique for suitable developer assignment. Recent studies have emphasized that time based decay is efficient in bug triaging. It is due to the fact that 'knowledge decays over time'. Thus, we propose and evaluate a novel time based model for bug report assignment. It has also been observed in literature that all the bug parameters used for bug report assignment has been given equal weightage whereas in the real scenario bug parameters can play role with varying importance. Hence, we propose a novel bug assignment approach, W8Prioritizer, based on parameter prioritization. We further extend our study for triaging of Non-reproducible (NR) bugs. Whenever the developer faces any issue in reproducing a bug report, he/she marks the bug report as NR. However, certain portion of these bugs gets reproduced and eventually fixed later. To predict the fixability of bug reports marked as NR, we propose a prediction model, NRFixer. We plan to work on bug report assignment for fixable NR bugs. Overall our initial results are encouraging and shows the possibility of making a robust recommender system for efficient bug report assignment for both reproducible (R) and NR bugs.

Keywords. Bug triaging, Bug report assignment, Recommender systems, Mining software repositories.

1 Introduction

Software bugs are inevitable and bug triaging is a difficult and time consuming task. Bugs are the programming error that causes significant performance degradation. They induce poor user experience and low system throughput. Large software projects use bug tracking repositories (or issue tracking systems) to collect, organize and keep track of all the reported bugs. The users, developers and testers all report the bugs they encounter to the bug repositories where these bugs are further analyzed by bug triager to verify the existence of bug. One of the main challenges bug triager faces is to select the

most competent developer for bug report. The choice of developer is often based on his past activities (or interest areas). Various bug triaging techniques exist in literature. Previous studies show that optimizing bug triaging is a non-trivial activity and bug triager often faces difficulty in it. Hence, bug triaging techniques that can help the triager in making strategic decision can be beneficial.

In this research, we first perform a systematic literature review of existing bug triaging techniques to gauge the current trends in bug report assignment. In addition, we perform in-depth study to analyze the effect of popular bug triaging techniques on the efficiency of bug report assignment. Moreover, it has been found that existing bug triaging approaches use different parameters for developer assignment. Certain approaches use textual parameters while others use bug meta-field parameters. We perform a study to identify the best parameters among meta-fields, textual contents and amalgamation of meta-fields and textual contents. One of the important factors that plays an integral role in developer selection is recency (or time of usage). This is due to the fact that accuracy of developer knowledge is statistically correlated with time. Recent studies have also emphasized that time based decay is efficient in bug triaging [1-3]. The existing studies have used bug textual features with time decay for bug assignment. Since bug meta-fields are found to be most suitable parameters from the study, we proposed a novel bug meta-field oriented time decay based model [4] for bug report assignment.

Past studies propose varied bug report assignment approaches considering all input parameters on same platform. Hence, currently there is no bug triaging approach that gives bug parameters varying priorities. In real time scenario, bug parameters can play a role with varying importance in decision making. Hence, we use phenomenon of parameter prioritization in bug triaging. Analytic hierarchy process (AHP) is a technique for decision making that involves parameter prioritization. We propose an AHP based bug triaging technique, W8Prioritizer, to optimize the efficiency of bug report assignment technique.

Further, we foster a bug assignment model for a special category of bugs known as NR bugs. Ideally, a bug report should provide enough knowledge for developers to reproduce and fix the issue. However, reproducing some bugs is difficult. When the developer's all efforts to reproduce the bug fails, he or she marks the bug as NR. Some NR bugs are reopened in future and are marked as fixed. The fixation of NR marked bugs prompts a question on the creativity, productivity and quality of developers who previously marked the bug reports as NR. A prediction model to evaluate the probability of fixation of NR bug can be beneficial as this will save time utilized on those NR bugs whose probability of fixation is negligible [5]. Thereafter, a bug triaging technique for fixable NR bugs can be useful for solving the NR bug.

The main contributions of this research will be development of a proficient recommendation system for bug report assignment. We illustrate bug triaging through several quantitative and qualitative models. In essence, these models apply time decay and parameter prioritization for bug report assignment process. We further use these models to accomplish appropriate developer recommendation for NR bugs. Our intuition is that before developer selection for NR bugs if there exists a prediction model that could

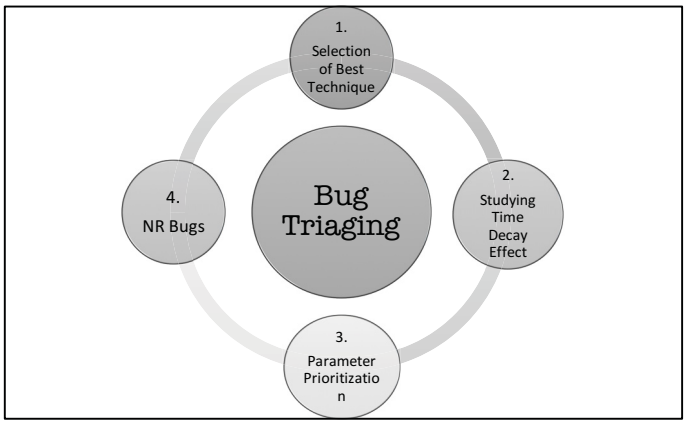


Fig. 1. Major components of bug triaging studied in this work

evaluate the fixability of NR bugs then it will be advantageous for both bug triagers and developers. With the usage of such model, software developers can easily dedicate their valuable time and efforts only on those bugs that have excessive probability of getting fixed and are observed as fixable by the proposed mechanism. Figure 1 shows the four major components related to bug triaging studied in this work.

2 Related Work

Several researchers have proposed different bug assignment approaches to semi or fully automate the bug triaging process. Various approaches considered bug report assignment as a text classification problem [6-8]. Using supervised learning, Cubranic et al. [6] classified 30% of bug reports correctly. Naguib et al. [9] proposed an information retrieval (IR) based technique for developer recommendation. An activity profile is generated for each developer based on the activities performed by him in the past. The profile generated is the indication of knowledge and expertise of the developers. The final ranking of developers corresponding to a new bug report is done according to profile generated for the users. The approach is tested on three software projects, Eclipse, UNICASE and ATLAS Reconstruction and can obtain the average hit ratio of 88% for top-10 recommendation list. Similarly, various other studies also use IR based techniques [1-3]. Bhattacharya et al. [10] proposed the technique of using tossing graphs for bug report assignment. They integrated the concept of using tossing graphs with machine learning techniques. They concluded naïve bayes to be the best classifier for bug report assignment. Hosseini et al. [11] proposed an auction based mechanism for developer selection in which whenever any new bug report arrives, the bug triager auctions it off and collects all the requests from different developers.

Certain recent studies utilized association rule mining [12], optimization techniques such as genetic algorithms [13] for bug report assignment. All these techniques consider different features with same priority to make the decision but often a decision

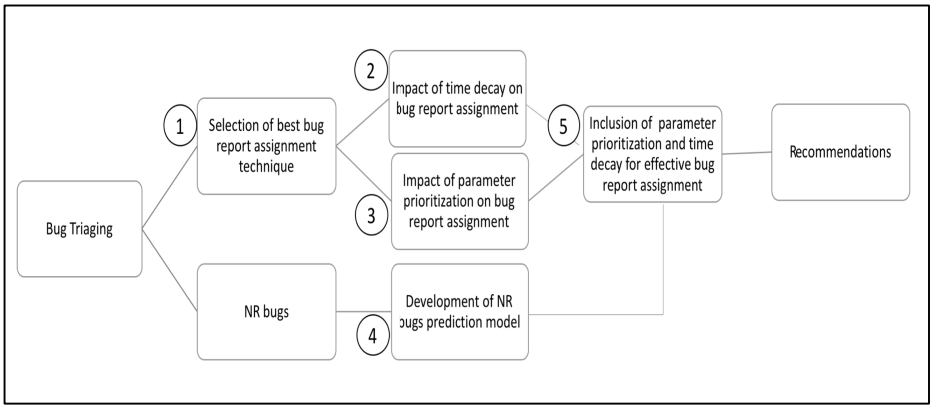


Fig. 2. Illustration of five step study to obtain efficient bug triaging.

is based on multiple criteria bearing different weights (or priorities) among each other. Panagiotou et al. [14] proposed STARDOM (Software Developer competency profiler) which builds an activity profile of developers working on a project in bug repository. They computed fluency, contribution, effectiveness and recency as their features. Further, AHP is used to prioritize the features and make the developer recommendations

In context to NR bugs, Joorabchi et al. [15] presented an empirical survey of NR bugs. They reported that 17% of all the reported bugs are marked as NR. Out of these, 3% are later fixed and 45% gets fixed implicitly. They also found that compared with other bugs, NR bugs remain open for three months longer. Shihab et al. [16] studied the nature of bugs that gets reopened. Although reopening a bug increases maintenance costs and leads to unnecessary rework by busy developers but they may get fixed as well. They build a decision tree using various factors that aims to predict reopened bugs. Though the factors that best indicate reopened bug vary based on the project, the keywords generated from the comment text was found to be the most important factor that impacts bug reopening. Guo et al. [17] performed an empirical study to characterize the factors that affect which bugs get fixed. They found that people who have been successful in getting their bugs fixed in the past are more likely to get their bugs fixed in future. Garcia et al. [18] further analyzed the prediction of blocking bugs. They used fourteen meta-field factors to build the prediction model which achieved an F-measure of 15-42% for predicting whether a bug would be blocking bug or not.

3 Research Questions

We explore the following main research objectives (RO) in this work:

RO1: Perform in-depth study of bug triaging techniques.

RO2: Build a time-based model for bug triaging.

RO3: Build bug triaging model based on parameter prioritization.

RO4: Analyze and build a prediction model for NR bugs.

The comprehensive view of the entire research work is presented in figure 2.

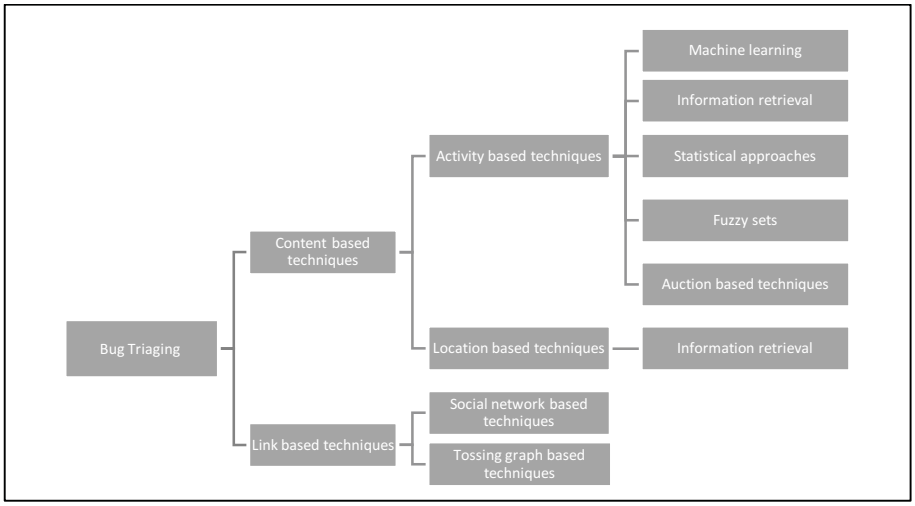


Fig. 3. Classification of bug assignment approaches.

4 Proposed Solutions and Results

In the initial part of this research, we focused on investigating the effect of different factors such as time decay and parameter prioritization on bug report assignment. Besides these tasks, we develop a prediction model for NR bugs to find the possibility that a bug report, currently marked as NR, will get fixed in future or not.

4.1 RO1: Analysis of Existing Bug Triaging Techniques

In the last two decades, researchers have addressed the problem of bug report assignment exhaustively. Cubranic et al. [6] proposed one of the few initial semi-automatic bug assignment approach. They considered the triaging as a text classification problem. Since then various approaches have been proposed by different researchers and practitioners. These approaches could be broadly classified into activity based and location based techniques. We started our initial investigation related to bug report assignment by analyzing the available techniques in the literature. We performed a systematic literature survey of papers published in repute journals and conferences between the years 2004 to 2016 [19]. We identified subcategories under activity and location based techniques as the result of exhaustive survey. The sub categories are machine learning, information retrieval, statistical approaches, fuzzy sets, auction based approaches, social network and tossing graph based approaches. Figure 3 shows the classification of bug assignment techniques. From the systematic survey, it has been found that machine learning and information retrieval based approaches are most popular in literature. Further, a trend analysis of the popular techniques shows that current trend is taking a shift from machine learning based approaches to information retrieval-based approaches. We performed a comparative study to investigate the reason behind this shift [20].

Results: For experimental evaluation, we consider the bug reports of Mozilla, Eclipse, Gnome and OpenOffice projects in Bugzilla repository. We consider bug id, component, severity, priority, operating system and assigned-to fields of the bug reports. We collected a total of 59,448 bug reports with fixed resolution. For machine learning techniques, we use Naïve Bayes, J48, Random tree and Bayes Net algorithms. In context to IR based technique, we consider expertise calculation using term frequency technique. We create a term - author – matrix for all the unique terms and developers in the training dataset. All the values in the various meta-fields of bug report are considered as terms and all the unique developers are considered as authors. Each entry in the matrix represents the frequency of term with respect to a particular developer. The frequency represents the expertise of developer with respect to a term based on the work done by the developer in the past. Comparing the results of machine learning and information retrieval based techniques, we found that information retrieval based techniques outperforms machine learning based algorithms and thus is more suitable for activity profile based bug assignment approaches. This is due to the fact that IR based techniques consider the overall expertise of developers towards bug reports for developer recommendation. This leads to formation of a more efficient and realistic recommender system. Also, these techniques are easy to comprehend with various new techniques such as fuzzy sets, social networks, etc.

IR based technique is more effective than ML techniques for bug assignment.

4.2 RO2: Time Based Model for Bug Triaging

Shokripour et al. [1] proposed a time-based approach for automatic bug assignment. They emphasized that knowledge decays over time and thus the computation of expertise of developers should also constitute time as the factor for normalization. This inclusion lowers the weight for terms that were used earlier and keeps the training data up-to-date. We performed an empirical study [4] related to time based decay to quantify the effect of recency (or time of usage) on the efficiency of bug report assignment. We proposed a novel time based bug triaging model, Visheshagya, that considers various bug meta-fields for bug report assignment. In addition to existing bug parameters, we extracted the last changed date of bug reports and calculated the difference in time between the last usage date of term by developer and the current date of assignment. This calculated time factor is then used to normalize the frequency values in term-author-matrix, i.e. the time-based expertise is calculated by dividing all the frequency values of each developer in the term-author-matrix by their associated time factors. For example, if r is the current date of new bug assignment and c (t, d) is the date of last usage of term, t by the developer, d. Then expertise of developer, d with term, t can be calculated as:

$$Expertise(d,t) = \frac{f}{r - c(t,d)} \quad (1)$$

where, *f* represents the frequency of usage of term *t*, by developer, *d* in the past.

Results: The proposed model, Visheshagya, is being evaluated for bug reports of Mozilla and Eclipse projects. The information retrieval based technique which is found to be effective in activity profiling of developers is applied to evaluate the effect of time decay. We compared non-time based and time based weighting approach for bug report assignment and found that time decay based techniques help in obtaining better efficiency. We are still investigating which time degradation measure (days, months or years) is most suitable for bug report assignment as different researchers used different measures in their evaluations.

Inclusion of time based decay in expertise calculation of developers increases the efficiency of IR based technique.

4.3 RO3: Bug Triaging Based on Parameter Prioritization.

Bug report assignment approaches extract bug parameters from the historically fixed datasets and creates corpus which is later used for suitable developer selection for new bug report. Researchers have augmented the use of different bug report parameters for corpus creation. In addition to diversity of parameters used for bug triaging, various studies have concluded different bug report parameters to be more important than others for bug triaging. Thus, different parameters should be weighed differently according to their priority for bug report assignment. Hence, we propose an AHP based technique, W8Prioritizer, for developer selection in bug repositories. AHP assigns the priorities to the bug report parameters to highlight the parameter importance before developer selection. It is a popular technique for multi criteria decision making which is a sub discipline of operational research that explicitly considers multiple criteria for decision making [21]. AHP is used in numerous situations that includes ranking, prioritization, selection, etc. We propose the usage of AHP based criteria prioritization method for prioritizing the various parameters of bug report and to obtain optimization in developer selection for bug triaging. In the proposed approach, we create the activity profiles of developers based on their past commits. We further build a matrix of the pair wise comparison ratings to determine the priorities for all bug parameters. Finally, the developer activity profiles (in term-author-matrix) are synthesized according to these parameter priorities. Now whenever a new bug report arrives, its tokens are extracted and the developers with maximum expertise towards new bug report tokens are selected to resolve the bug.

Results: The parameter prioritization based approach, W8Prioritizer, achieved an improvement of 20.59% and 38.57% in accuracy for Mozilla and Eclipse projects respectively. We further plan to include time based degradation factor in AHP based bug assignment approach.

Parameter prioritization helps in optimized developer selection for bug report assignment.

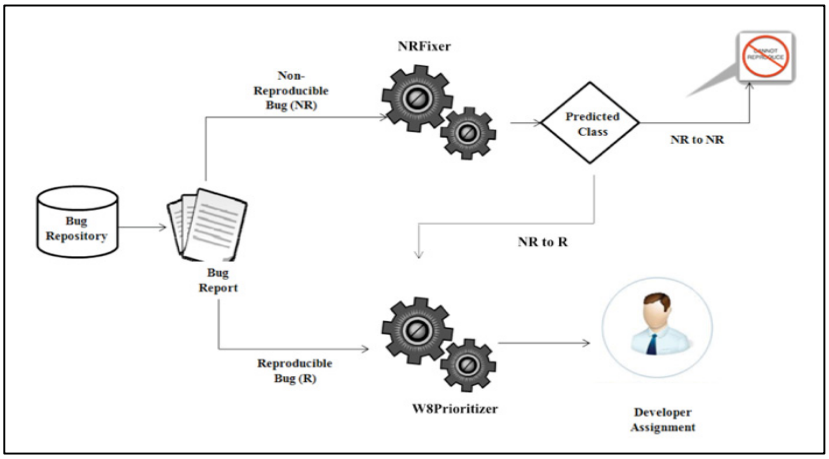


Fig. 4. Framework for optimized bug triaging.

4.4 RO4: Bug Triaging for Non-Reproducible Bugs

NR bugs account for approximately 17% of all bug reports and 3% of these bugs are later marked as fixed [15]. There could be various reasons behind this fixation of NR bugs. This may be due to any new code patch that might be made available by the reporter, user or developer which could help to reproduce the cause of bug, or there may be various solutions to fix a bug. Thus, the choice of solution tried by the developer to reproduce or fix the bug could be wrong [22]. Another reason could be that the developer had initially marked the bug as NR erroneously due to negligence or may be, in reluctance to reduce his or her workload. If we can have a mechanism that can provide information to developer beforehand that the bug report currently marked as NR will be fixed in future or not, it will not only provide insights to triager but also helps developers by predicting the possibility that whether bug report marked as NR could get fixed in future or not. This will save time, effort and cost incurred in those NR bugs which have less probability of getting fixed. With the use of such mechanism, developers & triager can devote their precious time and efforts on those bugs that are regarded as fixable by the proposed mechanism. This will also raise the level of interest among developers towards NR bugs. Thus, we are developing a prediction model, NRFixer [5] to predict the probability of fixation of bug report currently marked as NR. The NR bug reports predicted as fixable by prediction model will be assigned with a new developer using the proposed bug assignment technique who will try to reproduce the bug and may fix it.

Results: The proposed prediction model, NRFixer has been evaluated on Mozilla and eclipse bug reports and achieves precision value up to 74.7% for Mozilla bug reports and 68% for eclipse bug reports.

It is possible to predict whether the bug report marked as NR will get fixed in future or not.

5 Current Status & Future Plan

Till now, we have implemented several empirical investigations related to bug triaging. A comparative analysis of popular techniques used for bug triaging has been conducted to discover the most appropriate procedure for bug triaging. We implemented a bug assignment approach using an additional time degradation factor. We also proposed and evaluated a parameter prioritization based bug assignment approach. The experimental results show that both time based knowledge decay and parameter prioritization helps in building more precise developer recommendation models individually. We further proposed NRFixer, a prediction framework to predict the fixability of bug reports marked as NR.

In the future, we first plan to integrate the parameter prioritization model with the time decay model. Second, we plan to evaluate the effectiveness of bug report assignment for NR bugs that are predicted as fixable by NRFixer. The overall framework for this research work is presented in figure 4.

References

1. Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2015). A time-based approach to automatic bug report assignment. Journal of Systems and Software, 102, 109-122.
2. Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2014). Improving automatic bug assignment using time-metadata in term-weighting. IET Software, 8(6), 269-278.
3. Matter, D., Kuhn, A., & Nierstrasz, O. (2009, May). Assigning bug reports using a vocabulary-based expertise model of developers. In 2009 6th IEEE International Working Conference on Mining Software Repositories (pp. 131-140). IEEE.
4. Goyal A., Mohan, D., & Sardana, N. (2016). Visheshagya: Time based expertise model for bug report assignment. In Contemporary Computing (IC3), 2014 Seventh International Conference on (pp.1-6). IEEE.
5. Goyal, A., & Sardana, N. (2017). NRFixer: Sentiment Based Model for Predicting the Fixability of Non-Reproducible Bugs. e-Informatica Software Engineering Journal, 11(1), 109-122.
6. Cubranic, D. & Murphy, G. (2004). Automatic bug triage using text categorization. In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering.
7. J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," In Proc. 22nd Int. Conf. on Software Eng. and Knowledge Eng., SEKE '10, pp. 209-214.
8. Anvik, J., & Murphy, G. C. (2011). Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Transactions on Software Engineering and Methodology (TOSEM), 20(3), 10.
9. Naguib, H., Narayan, N., Brügge, B., & Helal, D. (2013, May). Bug report assignee recommendation using activity profiles. In Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on (pp. 22-30). IEEE.
10. Bhattacharya, P., Neamtiu, I., & Shelton, C. R. (2012). Automated, highly-accurate, bug assignment using machine learning and tossing graphs.Journal of Systems and Software, 85(10), 2275-2292.

11. Hosseini, H., Nguyen, R., & Godfrey, M. W. (2012, March). A market-based bug allocation mechanism using predictive bug lifetimes. In Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on (pp. 149-158). IEEE.
12. Sharma, M., Kumari, M., & Singh, V. B. (2015, June). Bug Assignee Prediction Using Association Rule Mining. In International Conference on Computational Science and Its Applications (pp. 444-457). Springer International Publishing.
13. Karim, M. R., Ruhe, G., Rahman, M., Garousi, V., & Zimmermann, T. (2016). An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. Journal of Software: Evolution and Process.
14. Panagiotou, D., & Paraskevopoulos, F. (2011, March). Specifications of developer profile.
15. Erfani Joorabchi, M., Mirzaaghaei, M., & Mesbah, A. (2014, May). Works for me! Characterizing non-reproducible bug reports. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 62-71). ACM.
16. Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., ... & Matsumoto, K. I. (2013). Studying re-opened bugs in open source software. Empirical Software Engineering, 18(5), 1005-1042.
17. Guo, P. J., Zimmermann, T., Nagappan, N., & Murphy, B. (2010, May). Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In Software Engineering, 2010 ACM/IEEE 32nd International Conference on (Vol. 1, pp. 495-504). IEEE.
18. Valdivia Garcia, H., & Shihab, E. (2014, May). Characterizing and predicting blocking bugs in open source projects. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 72-81). ACM.
19. Goyal, A., & Sardana, N. (2016). Analytical Study on Bug Triaging Practices. International Journal of Open Source Software and Processes (IJOSSP), 7(2), 20-42.
20. Goyal, A., & Sardana, N. (2017). Machine Learning or Information Retrieval Techniques for Bug Triaging: Which is better?. e-Informatica Software Engineering Journal, 11(1), 123-147.
21. Triantaphyllou, E., Shu, B., Sanchez, S. N., & Ray, T. (1998). Multi-criteria decision making: an operations research approach. Encyclopedia of electrical and electronics engineering, 15, 175-186.
22. Murphy-Hill, E., Zimmermann, T., Bird, C., & Nagappan, N. (2015). The Design Space of Bug Fixes and How Developers Navigate It. Software Engineering, IEEE Transactions on, 41(1), 65-81.

Propagation of Requirements Engineering Knowledge in Open Source Software Development: Causes and Effects – A Distributed Cognitive Perspective

Deepa Gopal[1] and Kalle Lyytinen[2]

[1] Case Western Reserve University, Ohio, USA
deepa.gopal@case.edu
[2] Case Western Reserve University, Ohio, USA
kalle.lyytinen@case.edu

Abstract. Popularity of open source software (OSS) development projects has spiked an interest in requirements engineering (RE) practices of such communities that are starkly different from those of traditional software development projects. Past work has focused on characterizing this difference while this work centers on the variations in the propagation of RE knowledge among different OSS development endeavors. The OSS RE activity in OSS communities is conceptualized as a socio-technical distributed cognitive (DCog) activity where heterogeneous actors interact with one another and structural artifacts to 'compute' requirements. These coordinated sequences of action are continuously interrupted and shaped by the demands of an ever-changing environment resulting in various DCog configurations and are visible in the communicative pathways deployed by the communities. We explore how the DCog configurations in OSS communities manifesting the flow of RE knowledge respond to the attributes of the environment housing the projects and their effects on the attributes of software requirements produced by such communities. The requirement attributes are measured using a 6-V requirements model centered on the volume, veracity, volatility, velocity, vagueness and variance of software requirements while the DCog configurations of RE knowledge flow is measured using social network analysis of the requirement activities in OSS projects. It is hypothesized that low communication centrality in OSS communities is more effective in task completion while facing a higher volume and velocity of requirements from its environment. Lower communication centrality is also hypothesized to result in more veracious and less vague software requirements produced by its members. The hypotheses of the study are tested using a mixed method methodology including a qualitative comparative case study and a quantitative analysis of selected sample of SourceForge OSS projects.

Keywords: Open source software, requirements quality, distributed cognition, mixed methods, social network analysis, 6-V requirements model, communication centrality

1 Introduction

The determination and management of system requirements continues to be one of the major challenges of contemporary software development [4]. One challenge that recently has confronted researchers is how to characterize the determination of requirements in non-traditional contexts, such as Open Source Software (OSS). In this context, classical requirements artifacts and processes are almost completely absent [1] and are accomplished through the use of 'informalisms' (such as bug trackers and discussion forums [38]. It has also been argued that requirements engineering (RE) in OSS is a high level distributed cognitive (DCog) process spread over time and space comprising of multiple stakeholders and heterogeneous artifacts [13]. All in all, past work has mainly focused on delineating the features which make OSS RE distinct from RE in traditional forms of software development.

However, OSS is not a unitary form of software development. It is that the social structures of OSS projects are a function of its scope [5] which can be manifested in the number of requirements faced by the respective OSS communities and the rate at which such requirements change over time. Given these observed variations it is unlikely that requirements are determined in a unitary fashion across all OSS projects. Given that certain environmental factors like volume of requirements affect the way RE is conducted in OSS communities ultimately impacting the RE quality and success of such projects, more research is needed in understanding what causes the RE variations. To characterize differences in RE practices across various OSS projects a DCog view of OSS [13] is deployed. This view is sensitive to the dynamic and distributed nature of practices in the OSS context, and assumes that multiple actors deploy heterogeneous artifacts to 'compute' requirements to reach a common understanding of what the software is going to do. These coordinated interactions are manifestations of the RE knowledge propagation and the sequences are continuously interrupted and shaped by its environment [30]. Further, drawing upon the Information Processing View (IPV) [9] [10], it is conceptualized that the various ways in which an OSS community organizes its cognitive activities socially and structurally is a response to the RE environment or more specifically, to requirements emanating within the environment [19]. These diverse DCog configurations in turn affect the quality of software requirements produced by such communities [14]. To understand the reciprocal relationship between the varying attributes of requirements that are addressed by different configurations of DCog activity for 'computing' requirements, the following question needs to be addressed:

- How can the DCog mode of each community be explained by the attributes of requirements emanating from the environment?
- How is the quality of requirements thus produced affected by its DCog configuration?

2 Literature Background

2.1 Requirements Engineering in Open Source Software

So far only a sparse amount of studies have shed light on the RE activities of OSS groups [37]. They have established that RE processes in OSS communities are starkly different from those in traditional software development due to the voluntary nature of participation in OSS development [5] and the use of informal web-based documentation practices which replace formal specifications and other design documents [31] [32]. The requirements in OSS projects are made explicit through a wide range of 'informalisms' such as threaded discussion forums, web pages, e-mail communications, and external publications [31]. Accordingly, OSS RE is considered to be less formal and dependent on online documentation and communication tools [7] [27]. Though this research provides detailed explanations of how distributed artifacts support RE, it does not consider the flow of requirements computation through interaction of actors and artifacts. A model of this interaction is suggested by [35]. They opine that the quality of RE is related to the structural distribution of the OSS project and the use of diverse artifacts through which requirements knowledge is disseminated. A recent study [38] suggests moreover that OSS RE is a socio-technical DCog activity where multiple actors deploy multiple artifacts to compute requirements to reach a common understanding of what the software is going to do. The organization of developer communities demonstrates significant variation around the generic core-periphery model [26] suggesting that OSS projects exhibit considerable diversity in their social and structural distributions. However the reason behind this diversity and how it is reflected in RE activity remains a largely unexplored area.

2.2 Distributed Cognition

To accommodate the distributed nature of OSS wherein requirements knowledge is distributed through multiple actors, artifacts and their interaction the DCog theory [16] [18] is used as the theoretical lens of inquiry. The theory postulates that cognition is not limited to mental states in the skull of an individual but rather it is deeply distributed among the social actors and artifacts which together constitute a system. Cognition is perceived as a socially and structurally distributed phenomenon where cognitive workload is shared among the members of a team and its artifacts [15] [17]. It is also fit to examine RE in OSS through this lens as it involves multiple actors and heterogeneous artifacts and complex cognitive processes.

Within this framework, cognitive activities are viewed as computations which take place via the propagation of representational state across media where the media can be both internal (e.g. individual memories) and external representations (including both computer and paper-based displays) [30]. The knowledge propagation can be characterized by communicative pathways deployed in such communities, conceptualized as coordinated sequences of action that are continuously interrupted and shaped by the demands of an ever-changing environment [30]. These cognitive processes are distributed across the members of a social group; and involve coordination between internal

and external (material or environmental) structure. Furthermore, processes can be distributed over time in such a way that the products of earlier events will transform the nature of later events [16]. These three forms of distribution have been identified as 'social distribution' (the distribution of cognition among actors), 'structural distribution' (the distribution of cognition across artifacts), and 'temporal distribution' (the distribution of cognitive process and tasks over time) [13] [35]. In the context of OSS RE, it is opined that the RE tasks of discovery, specification and validation evolve in parallel and through iterative loops as requirements are continuously refined and computed by the interaction of the social actors and structural artifacts over the lifetime of an OSS project [38]. These otherwise 'invisible' patterns of interactions which characterize the particular DCog mode present in an OSS development community can be identified and assessed with the aid of social network analysis.

2.3 Social Networks in Open Source Software

A distributed context like OSS community has been presented as an example of a dynamic social network and also as a self-organizing collaboration network [23] that is complex and evolving [24] where developers collaborate with each other through the internet based platforms [39]. Compared to traditional organizations, OSS community in itself is highly decentralized [24], flat and non-hierarchical [3]. This view has been challenged lately [5] [26]. These scholars argue that the social structure of OSS communities is layered like an onion rather than flat [5] [8]. The communication structure of such communities has been found to vary from completely centered on one developer to projects that are highly decentralized and exhibit a distributed pattern of communication between core developers and other active users [5] thus implying that the DCog modes in these communities are not uniform.

Though it has been studied that the social DCog modes of OSS communities vary, the consequences of such variances are unknown. Given the variations in OSS social structures and the effects of such structures in project success, in the context of OSS RE, it is important to understand how the formation of those social structures influence knowledge sharing and maintenance and thereby drive a successful RE process. This looks into how variation in requirements knowledge is influenced and conditioned by the underlying DCog mode of the community exhibited in terms of its social network structure and warrants further academic investigation.

2.4 Influence of Requirements Emanating from the External Environment

To uncover how the different DCog modes can be explained by the environmental characteristics housing the OSS groups, the IPV theory is appropriate. IPV posits that managers use organizational mechanisms, such as communication flows and work processes, to address the information processing needs of the organizational tasks. Alternative organizational mechanisms are geared towards either reducing information processing needs or increasing capacity for processing information [9] [10]. The choice of the mechanisms is dependent on the amount of information that needs to be processed. The information processing needs in itself stem from the level of environmental uncertainty. Furthermore, in a context where the cognitive activity is distributed socially,

structurally and temporally, the communicative pathways exposing the underlying sequences of interactions between actors and artifacts are in response to the demands of the environment [30]. Thus in OSS RE, the propagation of RE knowledge characterized by its communicative pathways manifesting its DCog mode is a product of the demands of its environment. It can be inferred that the environmental characteristics of various OSS groups are likely to invoke varying DCog mechanisms depending on the type of RE task they need to address. The information processing needs are also related to the complexity associated with RE. The perception of this complexity has shifted from managing inner and static complexity (the set of requirements remains stable since its inception) to a dynamic external form of complexity (the set of requirements is dynamic and has high level of dependencies) [19]. The requirements thus emanating from the RE environment can be studied in terms of six V's – volume, veracity, vagueness, velocity, variance and volatility.

In this regard the design task is approached as an effort to improve the environmental 'fit' of the software system by adapting it into a growing number of technical, social and organizational subsystems [14]. Thus the DCog configurations 'chosen' by an OSS project can be seen as a direct response to the specific environmental factors it is subjected [12]. Along with the exact mechanism of such variations, its consequences are still unknown.

2.5 Factors Affecting Quality of Requirements Produced

Given the emphasis placed on quality of RE phase in information systems development, it is interesting to look at if and how various OSS DCog configurations affect the quality of requirements produced by such project development communities. Quality of requirements in general can be studied in terms of the atomicity, precision, completeness, consistency, understandability, unambiguity, traceability, abstraction, validability, verifiability and modifiability of requirements [11]. Though a rapidly changing environment is detrimental to the quality of RE, it has been found that user participation alleviates some of its negative effects [6]. The finding has been reinforced by a later study [21] that shows that user involvement is the key concept in the development of useful and usable systems and has positive effects on system success and user satisfaction. This insight is very valuable in determining the extent to which the stakeholders must be included in the RE phase of a project and especially in the OSS context, where participants are both producers and users of the end software product. The above findings emphasize the influence of social structure of development teams on the ensuing RE activity and in the OSS context, resounds in the manner in which the social distribution of DCog activities affect the RE process.

Effective communication is critical to the development of mutual understanding between systems professionals and their clients. It is seen that distributed RE is more effective when stakeholders, participate actively in synchronous activities of the requirements process. Three processes have been identified to help systems analysts establish mutual understanding with their clients: shifting perspective, managing transaction, and establishing rapport, that reflects an effective communication flow [34]. Intensive communication between the developers and customers is identified to be the

most important RE practice [2]. This finding is especially significant for OSS RE which is a dynamic process and exposes the effect of structural DCog activities on the RE process. Fostering mutual understanding and a shared vision among OSS development participants thus helps maintain a high quality of RE knowledge in the project and hence a high quality of RE in itself.

2.6 Impact of Communication Structures in Virtual Communities

Having understood how different environmental attributes contribute to different DCog configurations manifested by OSS communities and what factors influence requirements quality, it is important to explore what the outcomes, if any, of these DCog variations are in the RE process of OSS development projects. It is found that through close social interactions, individuals are able to increase the depth, breadth, and efficiency of mutual knowledge exchange [22]. In the context of software development, decentralized groups promote performance and creativity by enabling members to share knowledge in a more efficient and effective manner than centralized ones [20]. Thus in the context of OSS RE knowledge, decentralization of OSS communities enhance the quality of RE knowledge shared subsequently exhibiting higher RE knowledge quality highlighting how various DCog modes affect requirements quality of OSS communities.

The above literature review reveals that the environment housing an OSS project influences its configuration of DCog activity exhibited in its communication structures which in turn impacts the quality of requirements the project produces. However the requirements quality concepts dealt with in literature so far are from the perspective of managing the inner static complexity of design tasks [19]. To understand the interdependency between a changing environment that OSS projects are subject to and the RE outcomes of such development efforts, it is important to account for the external dynamic complexity of RE tasks which calls for the evaluation of the requirements stemming from the environment of OSS communities and quality of requirements they produce via the lens of the 6-V requirements model proposed by [19]. Thus the following theoretical model attempts to unveil the relationship between the external environment housing an OSS project, the DCog configuration it chooses and the quality of requirements it produces is studied in terms of the 6-V requirements model attributes and its social communication structure.

3 Theoretical Model

To address the effects between the environmental characteristics and the DCog configuration of an OSS community, the work on social structures of OSS projects [5] helps shed some light. The authors opine that the social structure of OSS communities is not constant with communication centrality varying form completely centralized on one developer to vastly decentralized. It has been hinted that OSS projects with a wider scope often take on a modular social structure and are decentralized [5]. This resonates with the IPV theory that postulates that when the environmental uncertainty is greater, an organized entity creates more self-contained tasks to reduce the need to process more

information and thereby obliterating the need to over-burden the hierarchy [10]. Thus, in an OSS context, it can thus be inferred that to remain effective in situations where the project scope widens, the OSS communities have to create more modular task structures by grouping themselves to smaller sub-projects within the main project mirroring the shallot structure proposed by [5] along with decentralizing the communication. Thus, the communication centrality can be more or less effective for OSS communities, depending on the project scope it is subjected to.

It is arguable that the scope of OSS projects increase with functionality it offers and changes in the technological context it is embedded in. The amount of functionality offered by the end OSS product is manifested in the volume of requirements it faces and the rate of change in technology it is based upon affects the focus of development activity which is explicit in the velocity of change in the requirements it faces. The extent to which the team completes identified work tasks (task completion) is an output effectiveness construct that has been used previously [36] and is adopted in this study as the OSS project effectiveness measure. As demonstrated in prior work [36], we calculate task completion as the percentage of tasks completed: (total requests – requests open)/total requests * 100. Thus the initial hypotheses of the study can be summarized as follows:
H1: Lower communication centrality of an OSS community results in more task completion under conditions of higher volume of requirements.
H2: Lower communication centrality of an OSS community results in more task completion under conditions of higher velocity of requirement changes.

To address the effects of various social structural DCog configurations on the quality of requirements produced, we refer back to the literature review above which exposes how decentralized communication structures further efficient and effective knowledge exchange and increased stakeholder participation which in the context of RE activities help increase mutual understanding and thus enhance the quality of requirements produced resulting in more unambiguous, concise, well-understood requirements. Stakeholder dialog is a pillar of the requirements development process [28]. As project teams experience difficulties and communication breakdowns during the process of acquiring, sharing, and integrating project-relevant knowledge, requirements or their analyses are forgotten resulting in different requirements concerning the same objects. A lack of shared understanding by OSS participants can occur when the communication is highly centralized around a few select members. This produces inconsistency, ambiguity, and incompleteness in requirements [28] that are represented by the vagueness and veracity features in the 6-V requirement model. Viewing this in the context of list of desirable features in requirements [11], such attributes signal a poor quality of requirements. Decentralization of communication mitigates this effect by promoting discourse and co-learning [36] resulting in mutual understanding. Thus, in an OSS context, communities exhibiting a lower degree of communication centralization point towards effective RE knowledge exchange and thus higher RE quality manifested by requirements high in veracity and low in vagueness. Thus, the next two hypotheses of the study can be stated as:

H3: OSS communities with a lower degree of communication centrality produce requirements that are more veracious than those produced by communities with a higher degree of communication centrality.
H4: OSS communities with a lower degree of communication centrality produce requirements that are less vague than those produced by communities with a higher degree of communication centrality.

4 Research Design

Understanding the inter-dependencies between requirement attributes and DCog configurations of OSS communities requires identifying macro level external environmental characteristics of the projects as well as a rich understanding of the local development practices of the projects which requires a mixed methods design. OSS contexts are suitable for such a method of inquiry as rich qualitative data and quantitative data that are available from the digital traces left by such projects in collaborative code-hosting repositories such as Github or SourceForge can be utilized in the proposed study. This study aims to create theory in an area where none currently exists. To aid this theoretical development, a quantitative analysis of a sufficient sample size of OSS projects testing the stated hypotheses followed by an independent longitudinal qualitative case study of an OSS project. This will help paint an overall picture of OSS RE that explains how the hypothesized relationships exist and change over time.

To understand how the external environment housing OSS projects influence its DCog mode which in turn has repercussions on the quality of software requirements it produces, we propose to examine the network centrality of communication during the bug–fixing process. To mitigate the disadvantages arising from data that has not been cleaned properly and to encourage a cumulative tradition in the field of OSS research, we turn towards datasets that are been used by multiple academics and studies. This leads us to the SourceForge Research Data Archive (SRDA) data hosted by the University of Notre Dame [25]. The criteria for our sampling will be based on the project characteristics of interest in our study. Since one of the main constructs in our study is communication decentralization, we will initially filter out those projects that are not active, are individual efforts, or that do not make bug reports available on the SourceForge Tracker system, which is the source of the bug data in SRDA. Since we are interested in team interactions and not individual projects we will further limit our study to projects that listed more than seven developers. To present a robust picture of the social network of the OSS projects we need sufficient interactions to construct the same [5] and thus will choose only those projects that have a minimum of 100 reported bugs. Considering the requirements attributes of interest in our study, these will be analyzed from the feature requests in the OSS projects [29]. Hence, we will further filter our sample to include only those projects that have 600 or more postings in their feature request forums.

In the second phase, we will study a successful OSS project, Bootstrap, in terms of the varying environmental factors affecting its social network configuration and resultant requirements quality over a period of time. The project is hosted in both Sourceforge

and Github, is successful, having had multiple releases and involve multiple categories of developers and is a frontend web design framework originally developed by employers at Twitter. The project which in 2014 had only two core developers and 600 plus peripheral members has currently expanded its core along with more releases and branches in the interim. We will study the evolution of the social network structure of Bootstrap from 2014 to 2017 as a result of the environmental factors it has been subjected to and its effect on its requirements quality, if any.

5 Contributions

From a theoretical perspective, the study through a mixed methods approach, highlights the importance of the interaction between human actors and structural elements in OSS development environments that affects the way requirements are processed during a project's lifecycle. It also sheds new insights on the effects of environmental factors on the DCog configurations exhibited by an OSS group that in turn affects the quality of requirements produced by the group. From a practice perspective, the framing of requirements-oriented activities as a DCog process influenced by both its external environment and its internal social environment can inform OSS trend-setters to the critical role played by different stakeholders as well as the structural artifacts within the system. In addition, the study by presenting a range of DCog modes and its relationship to requirements attributes for various environmental backgrounds can help OSS communities in choosing those governance structures that are conducive to producing requirements of high quality irrespective of the vagaries of the environment they would be faced with. This is a crucial step in guiding OSS project leaders engaging in social community and structural artifacts maintenance activities to espouse a social structural climate exhibiting a high quality RE impervious to the mandates of the greater external world. The proposed causes and effects of RE knowledge propagation can also be evaluated in agile-based development environments, traditional software development projects and Commercial Off-The Shelf (COTS) product development. The application to multiple environments can foster inductive theory-driven comparison and identification of computational solution that influences project success and failure rates, developer and user satisfaction, or perceived innovativeness of solutions [38].

References

1. Alspaugh, T. A., & Scacchi, W. 2013. "Ongoing software development without classical requirements," In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pp. 165-174.
2. Cao, L., and Balasubramaniam, R. 2008. "Agile requirements engineering practices: An empirical study." Software, IEEE 25.1, pp. 60-67.
3. Carley, K. and Ahuja, M., 1999. "Network structure in virtual organization," *Organization Science*, (10:6), pp. 741-757.
4. Cheng, B. H., and Atlee, J. M. 2009. "Current and Future Research Directions in Requirements Engineering," In *Design Requirements Engineering: A Ten-Year Perspective*,

K. J. Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. N. Robinson (eds.), Berlin, Germany: Springer, pp. 11–43.
5. Crowston, K., and Howison, J. 2005. "The Social Structure of Free and Open Source Software Development," *First Monday* (10:2).
6. El Emam, K., Quintin, S., & Madhavji, N. H. 1996. "User participation in the requirements engineering process: An empirical study," *Requirements engineering*, *1*(1), 4-26.
7. Ernst, N. A., and Murphy, G. C. 2012. "Case Studies in Just-In-Time Requirements Analysis," In *IEEE 2nd International Workshop on Empirical Requirements Engineering*, pp. 25–32.
8. Gacek, C. and Arief, B., 2004. "The many meanings of open source," *IEEE software*, (21:1), pp. 34-40.
9. Galbraith, J. R. 1973. *Designing complex organizations* (p. 150). Addison-Wesley Pub. Co.
10. Galbraith, J. R. 1974. Organization Design: An Information Processing View. *Interfaces*, (*4*:3), pp. 28–36. doi:10.1287/inte.4.3.28
11. Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., & Moreno, V. 2013. "A framework to measure and improve the quality of textual requirements," *Requirements Engineering*, *18*(1), 25-41.
12. Gopal, D., Lindberg, A., Lyytinen, K. 2015. "Attributes of Open Source Software Requirements – The Effect of the External Environment and Internal Social Structure," *Proceedings of the 49th Hawaii International Conference on System Sciences.*
13. Hansen, S. W., Robinson, W. N., and Lyytinen, K. J. 2012. "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering," In *2012 45th Hawaii International Conference on System Sciences*, pp. 5224–5233.
14. Hanseth, O., Lyytinen, K. 2010. "Design Theory for Adaptive Complexity in Information Infrastructures," *Journal of Information Technology* (25:1), pp. 1-19.
15. Hutchins, E. 1995. *Cognition in the Wild,* MIT Press, Cambridge, MA, pp. 408.
16. Hutchins, E. 2000. "Distributed Cognition." *Internacional Enciclopedia of the Social and Behavioral Sciences*.
17. Hutchins, E. and Klausen, T. 1996. "Distributed Cognition in an Airline Cockpit," In *Cognition and Communication at Work,* Y. Engestrom and D. Middleton (eds.), Cambridge University Press, New York, pp. 15-34.
18. Hutchins, E., and Lintern, G. 1996. *Cognition in the Wild*, MIT Press, Cambridge, MA.
19. Jarke, M., and Lyytinen, K. 2014. "Special Issue on Complexity of Systems Evolution: Requirements Engineering Perspective," *ACM Transcations on Management Information Systems*
20. Kidane, Y.H. and Gloor, P.A. 2007. "Correlating temporal communication patterns of the Eclipse open source community with performance and creativity," *Computational and mathematical organization theory*, (13:1), pp.17-27.
21. Kujala, S., Kauppinen, M., Lehtola, L., & Kojo, T. 2005. "The role of user involvement in requirements quality and project success," In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on* (pp. 75-84). IEEE.
22. Lane, P.J., Lubatkin, M. 1998. "Relative absorptive capability and interorganizational learning," *Strategic Management Journal*, (19:5), pp. 461–477.
23. Madey, G., Freeh, V., and Tynan, R. 2002. "The open source software development phenomenon: An analysis based on social network theory," In: *Proceedings of Americas Conference on Information Systems (AMCIS 2002)*, Dallas, US, pp. 1806–1813.
24. Madey, G., Freeh, V., and Tynan, R. 2004. "*Modeling the F/OSS Community: A Quantitative Investigation, Free/Open Source Software Development*," Edited by Koch, S., Hershey: Idea Publishing, pp. 203–220.
25. Madey, G. ed. 2016. The SourceForge Research Data Archive (SRDA). University of Notre Dame. (2016) <http://srda.cse.nd.edu/>

26. 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 and Methodology* (11:3), pp. 309–346.
27. Noll, J., and Liu, W.-M. 2010. "Requirements Elicitation in Open Source Software Development," In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development,* ACM Press, pp. 35–40.
28. Robinson, W. N., & Pawlowski, S. D. 1999. "Managing requirements inconsistency with development goal monitors," *IEEE Transactions on Software Engineering*, (25:6), pp. 816-835.
29. Robinson, W. and Vlas, R., 2015." Requirements Evolution and Project Success: An Analysis of SourceForge Projects," In *Americas Conference on Information Systems* (AMCIS2015).
30. Rogers, Y., and Ellis, J. 1994. "Distributed cognition: an alternative framework for analyzing and explaining collaborative working," *Journal of information technology*, (9:2), pp. 119-128.
31. Scacchi, W. 2002. "Understanding the Requirements for Developing Open Source Software Systems," *IEE Proceedings Software* (149:1), pp. 24–39.
32. Scacchi, W. 2009. "Understanding Requirements for Open Source Software," In *Design Requirements Engineering: A Ten-Year Perspective*, K. Lyytinen, P. Loucopoulos, J. Mylopoulos, and B. Robinson (eds.), Berlin, Germany: Springer, pp. 467–494.
33. Stewart, K.J. and Gosain, S., 2006. "The moderating role of development stage in free/open source software project performance," *Software Process: Improvement and Practice*, (11:2), pp.177-191.
34. Tan, M. 1994. "Establishing mutual understanding in systems design: An empirical study." Journal of Management Information Systems 10.4, pp. 159-182.
35. Thummadi, B. V., Lyytinen, K., and Hansen, S. 2011. "Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development," *Sprouts: Working Papers on Information Systems* (11).
36. Toral, S.L., Martínez-Torres, M.D.R. and Barrero, F., 2010. "Analysis of virtual communities supporting OSS projects using social network analysis," *Information and Software Technology*, (52:3), pp. 296-303.
37. Vlas, R., and Vlas, C. 2011. "A Requirements-Based Analysis of Success in Open-Source Software Development Projects," In *Proceedings of the 17th Americas Conference on Information Systems,* Detroit, Michigan
38. Xiao, X., Lindberg, A., Hansen, S., and Lyytinen, K. 2013. "'Computing' Requirements in Open Source Software Projects," In *The 34th International Conference on Information Systems* (ICIS 2013).
39. Xu, J., Christley, S. and Madey, G., 2006. "Application of social network analysis to the study of open source software," *The economics of open source software development*, pp. 205-224.

Supporting Open Source Communities to Foster Code Contributions through Community Code Engagements

Jefferson O. Silva

University of São Paulo, Institute of Mathematics and Statistics, São Paulo, Brazil
silvajo@ime.usp.br

Abstract. Open Source Software (OSS) communities depend on recruiting new contributors to remain sustainable. Many communities are trying to retain new contributors and promote contributions by means of community code engagements (CCE), which are software development arrangements that include summers of code. There is empirical evidence that CCEs can be potentially more attractive to newcomers than the self-guided contribution to OSS, since many are held by high-profile software companies. However, the literature only provides evidence for specific domains, not offering practical insights on how effective these engagements are. This research aims at investigating how OSS communities can take advantage of CCEs, by fostering code contributions and retaining contributors. To achieve our goal, we plan to employ a mixed-method approach, combining data from different sources. We believe that the results can be compiled into a model, which may support communities and CCE organizers on fostering code contributions and engaging students as contributors.

Keywords: Summer of Code · Community Code Engagement · Open Source Software · Newcomer · Motivation · Sustainability · Attraction · Visibility · Early Identification.

1 Introduction

Recruiting new contributors is a difficult endeavor to many open source software (OSS) communities, especially because newcomers may not be strongly committed, making them more susceptible to leave in face of minor adversities [1]. As newcomers' ties are fragile, the communities may have to engage in tactics to keep newcomers around until they learn how to contribute [2].

There is evidence that many OSS communities are joining community code engagements (CCE), not only to attract newcomers, but also to retain them [3]–[5]. CCEs are short-term software development arrangements that are becoming common in OSS, which include Summer of Code (SoC) internships. SoCs promote

software development by students. Examples include Google SoC (GSoC)[1], Rails Girls SoC[2], Open SoC[3], Julia SoC[4], and Ruby SoC[5]. When applying for participating in CCEs, many OSS communities express the expectation to increase both the projects' visibility and the number of contributors, as exemplified by the following excerpt:

> "*(...) Participating at GSoC will increase the visibility of Pharo project efforts (...) We expect also to bring more people into our community*"
> Source: http://lists.pharo.org/pipermail/pharo-users_lists.pharo.org/2016-February/024579.html

Many CCEs are held by high-profile companies such as Facebook, and Google, which can potentially be more attractive to newcomers than the volunteer self-guided contribution to OSS. CCEs are a recent phenomenon and represent an additional entity not yet comprehensively studied. The current literature on CCEs mostly provides evidence on retention and code contribution for OSS projects belonging to the scientific domain [4]–[6]. The study of Schilling *et al*. [6] is restricted to the KDE project. In addition, these studies only targeted GSoC.

The current literature does not offer adequate support for the OSS communities to decide if participating in CCEs is a good strategy to remain sustainable. In addition, to our best knowledge, the existing evidence on how much the students' code can be incorporated to the codebase is scant. Finally, little is known on how to help the OSS communities to select the CCE applicants focusing on increasing the odds of retaining the participants after the engagement program.

This research aims at understanding how to support OSS communities that want to take advantage of the participation in CCEs, and also at proposing and evaluating a model to assist these communities to rank and select promising candidates. The overall question addressed by this research is:

"How to support OSS communities to foster code contributions through CCEs?"

To guide our answer, we have defined specific research questions (RQ):

RQ1. Do CCEs foster code contributions to OSS projects?

RQ2. Do CCEs retain students as code contributors of OSS projects?

RQ3. What motivates students to enter CCEs?

RQ4. What barriers do OSS communities face when trying to incorporate the code produced in CCEs into the codebase?

RQ5. What factors influence the CCEs students' retention?

---

[1] https://developers.google.com/open-source/gsoc/

[2] http://railsgirlssummerofcode.org/

[3] http://2016.summerofcode.be/

[4] http://julialang.org/blog/2015/05/jsoc-cfp/

[5] http://www.rubysummerofcode.org/

In this thesis' proposal, we focus on studying how to support the OSS communities to take advantage of CCEs (*i.e.*, fostering code contributions to OSS and retaining the students as new contributors). We believe that the results of this study can help researchers and the OSS community to invest their efforts in building or improving tools, ultimately gaining more effective contributions from participants interested in short and long-term collaboration.

2 Related Work

Despite its practical relevance, little research has examined how CCEs influence participants to contribute. For online communities, the literature offers several models about volunteer contributions, such as the Feature Article process, where the idea is to provide visibility and concentrate efforts to provide high-quality content to the Wikipedia [7].

In OSS, typically, studies on retention take the individual developer's perspective. Thereby, intrinsic motivation (*e.g.*, [8]–[10]), social ties with team members (*e.g.*, [11]–[13]), project characteristics (*e.g.*, [14]–[16]), ideology (*e.g.*, [17]), and rewards (*e.g.*, [18]–[20]) have been found relevant for developers' retention.

Schilling *et al*. [6] used the concepts of Person-Job and Person-Team fit from the traditional recruitment literature to derive objective measures to predict the retention of 80 former GSoC students in the KDE project. Using a classification schema of prior code contributions to KDE, they found that intermediate and high levels of prior development were strongly associated with retention.

Trainer *et al*. [5] conducted a case study of a bioinformatics library called Biopython to investigate the outcomes of this GSoC project. By interviewing the top 15 developers ranked by the number of commits, the researchers identified three positive outcomes: the addition of new features to the codebase; training; and personal development. In addition, the researchers found that mentors faced several challenges related to the selection and ranking of applicants.

Trainer *et al*. [4] conducted a multiple case study of 22 GSoC projects in the scientific domain to understand the range of GSoC outcomes and the underlying practices that lead to these outcomes. They found that GSoC facilitated the creation of strong ties between mentors and students, and reported that 18% of the students (n=22) became mentors in subsequent editions.

While these previous works help to enlighten understudied aspects of CCEs, their scope is restricted to a few GSoC projects and mainly to the scientific software domain, and consequently their conclusions may not be representative of other engagements. Schilling *et al*. [6] mined software repositories for quantifying students' retention, but limited their analysis to the KDE project. Trainer *et al*. [4], [5] collected data through interviews. Although we understand the relevance of interviews for achieving the researchers' goals, their results only represented the

subjective students' perception on the students' retention. In addition, in Schilling *et al*. [6], it is not clear whether there was students' code that was not incorporated into the codebase and how the researchers handled it. We are not aware of any studies for the other CCEs.

We argue that CCEs have the potential of influencing newcomers' decision process, which is not predicted by existing theories. The Legitimate Peripheral Participation (LPP) theory has been usually embraced to explain how an individual engages in a community of practice, such as OSS [21]. The LPP theory, applied to OSS, predicts that future contributors get involved by observing before coding, then interacting with experienced members at the margin, in a process that culminates in the emergence of regular contributors. However, contributing to OSS by means of CCEs alters this process in significant manner: there is a contract binding the students to the OSS projects for a period of a few months. CCE students do not start at the margin, instead, they are individually guided – many times, sponsored – to become contributors, having the time to dedicate themselves to the project, potentially developing strong social ties to the community members. We claim that it is relevant to provide more research so that theories can be tested under new conditions.

3 Research Method and Preliminary Results

This section details our planed approaches to address our RQs. In addition, we present some preliminary results for the work accomplished so far.

RQ1. Do CCEs foster code contributions to OSS projects?

Motivation. As previously evidenced, there are OSS communities that expect to have their software issues addressed by participating in CCEs. Answering this RQ may potentially help these communities to manage their expectations about how much contribution (*i.e.*, code churn and commits) the OSS projects should get by participating.

Approach. *Data Collection*. For collecting and analyzing the students' code, we have been using the method depicted in Fig. 1.

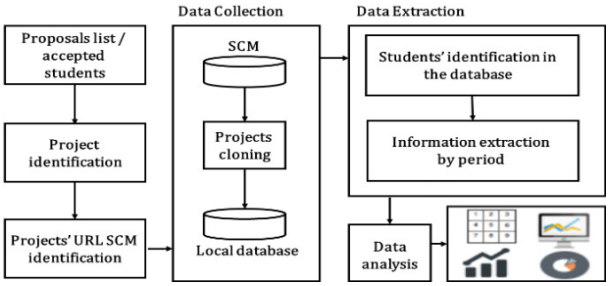


Fig. 1. Method used to collect and analyze the students' interaction with their GSoC projects**.**

We have started the investigations on a specific CCE by contacting the organizers, explaining our research goals, and eventually asking for the students' assigned projects URL's. As a rule, the organizers did not provide us with this information, probably for privacy concerns. Hence, we have obtained this information by consulting a list of accepted students and their assigned projects' names, which is usually publicly available online.

Next, as collecting and verifying data on the students and their assigned projects is time consuming, we have worked with a random sample of students for each CCE, aiming for a confidence level of 95% and a margin of error of 5%. Using the students' names and the projects' description contained in the lists, we have manually searched for the students' assigned projects in source code management systems (SCM).

We have used MetricsGrimoire-CVSAnalY[6] to extract information out of the SCMs and store it into a local database (one for each CCE). Next, we have to identify all the IDs that the students may have used. For this, we have used the students' names and emails (or combinations) to decide if an ID belongs to the same student. We have applied common disambiguation heuristics to identify the students, such as the ones presented by Wiese *et al*. [22]. This procedure has yielded the final working sample for each CCE.

Data Analysis. For the analysis, we have split the students' commits into three participation periods: *Before*, *During*, and *After* CCE. We have used the official engagements' timelines (*i.e.*, start and end dates) to classify the commits in each period. Then, we have analyzed the contributions in terms of: (*i*) commits; and (*ii*) code churn.

(*i*) *Commits' Analysis*. To understand if an OSS project benefited from the students' contribution, we have investigated whether the students' commits were incorporated into the codebase. Thus, we have compared each Secure Hash Algorithm (SHA) – a unique identifier – of the students' commits to the ones belonging to the main branch, and we have grouped them by participation period.

[6] http://metricsgrimoire.github.io/CVSAnalY/

The number of the students' commits in the codebase was obtained by counting the number of commits in each group.

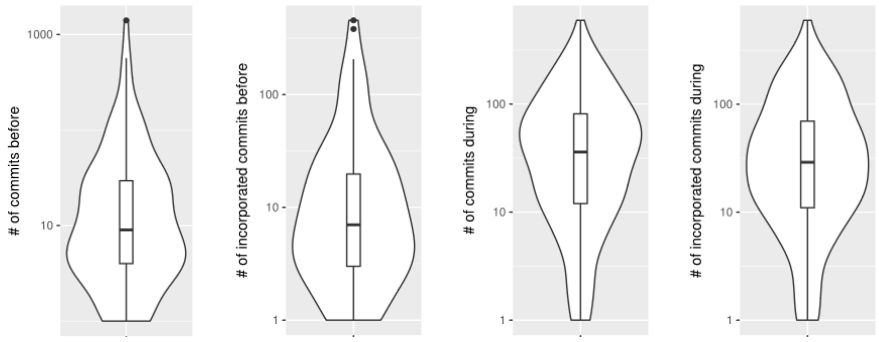
(*ii*) *Code Churn Analysis*. To estimate how much code the students added to the codebase, we used the following procedure. For each student, we used three tuples of commits' SHAs. The first tuple comprised the SHA of the first commit in the *Before* period and the SHA of the first commit in the *During* period. We represent this as (*SHA-first-before*, *SHA-first-during*). The other tuples, were obtained analogously: the second tuple (*SHA-first-during*, *SHA-last-during*); and the third, (*SHA-last-during*, *SHA-last-after*). Next, we used the git guilt (from git extras[7]) tool to calculate the code churn, using each tuple as arguments. As a result, we have obtained a measure for the churn for each period for each CCE.

Preliminary results. We decided to firstly investigate GSoC as it is a worldwide, consolidated Google program that offers students a stipend to write code for OSS for a three-month period. Also, it is well known compared to other SoCs, it has been in operation for more than 10 years, it has students from all over the world, and it provides students with a richer set of participation rewards than other SoCs [23], which, at least theoretically, may be attractive to students.

GSoC Sample Characteristics. We worked with a sample of 260 students: 75, in GSoC 2015; 62, in GSoC 2014; 56, in GSoC 2013; 12, in GSoC 2015 and GSoC 2014; 14, in GSoC 2014 and GSoC 2013; and 5, in GSoC 2013 to GSoC 2015. We first present: the (*i*) *Commit Analysis' Results*; followed by the (*ii*) *Code Churn Analysis' Results*.

(*i*) *Commit Analysis' Results*. To understand how many of the GSoC students' commits were incorporated into the codebase, we present the violin plots in Fig. 2 per participation period. For a better visualization of the data, we removed the students without commits in the plots. We report how many students were removed, in brackets.

[7] https://github.com/tj/git-extras



(a) Commits' distribution before GSoC [*130/260 (50%) did not commit*]

(b) Incorp. commits' distribution before GSoC [*158/260 (~61%) did not have any commit incorporated*]

(c) Commits' distribution during GSoC [*0/260 (0%) did not commit*]

(d) Incorp. commits' distribution during GSoC [*59/260 (~23%) did not have any commit incorporated*]

(e) Commits' distribution after GSoC [107*/260 (~41%) did not commit*]

(f) Incorp. commits' distribution after GSoC [*157/260 (~60%) did not have any commit incorporated*]

Fig. 2. Commits and incorporated commits distribution by participation period (*Before*, *During*, and *After*).

Fig. 2 (a) and Fig. 2 (b) show that there were both commits and code incorporated before kickoff, which may have come from at least three distinct sources: students who were already project members; former GSoC students; and newcomers. **One possible explanation for newcomers contribute before kickoff is that they contribute to OSS as a means to be accepted**.

Fig. 2 (c) and Fig. 2 (d) show that incorporations usually occurred during GSoC's timeframe, ranging from 1 (*first quartile*) to 92.50 (*third quartile*), totaling ~2,640 incorporated commits. During this timeframe, GSoC students performed 4,250 commits. In the worst cases (~25%), the OSS communities had not any incorporated commits, even though these students performed 2,875 commits. **Strictly, most OSS projects benefited from participation in GSoC since in 75%**

of the cases during the program they had at least one incorporated commit to the project codebase.

Fig. 2 (e) shows that 59% of the students committed after GSoC, and Fig. 2 (f) shows that ~40% of them had their commits incorporated to the codebase after the program. **Therefore, in all periods, there were code incorporations to the codebase**.

(*ii*) *Code Churn Analysis' Results.* We broadened our investigation by studying how much code the students added to the codebase. We used the concept of code churn to represent another perspective of the students' contribution.

Fig. 3 depicts the distribution of students' code churn in boxplots per participation period. The *churn before* boxplot shows that the distribution median is 913, with its top 25% ranging between 5,000 (*third quartile*) and 12,000 (*upper quartile*). For the next period, the distribution median approximately doubled (~1,900), presented in the *churn during* boxplot, with its top 25% ranging between 7,000 (*third quartile*) and 17,000 (*upper quartile*). The *churn after* boxplot shows that the students' code churn significantly decreases after the end of the program, with the distribution median decreasing to 8.5. However, the top 25% of the distribution remained high, ranging between 833 (*third quartile*) and 1,971 (*upper quartile*).

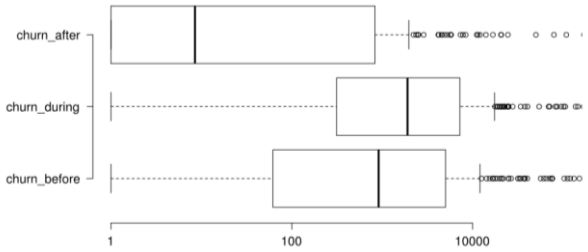


Fig. 3. Students' code churn by participation period.

Fig. 3 depicts the magnitude of the students' code contribution to the codebase: **the code churn Before GSoC totaled 21,599,122 (mean: 913, STD: 612,793.92), During, 28,388,710 (mean: 1905, STD: 687,706.04), and After, 5,863,721 (mean: 2, STD: 127,853.65).**

RQ2. Do CCEs retain students as contributors of OSS projects?

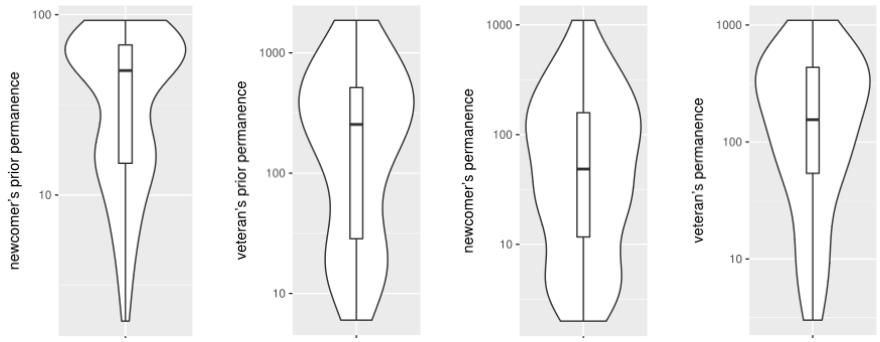
Motivation. Answering this RQ may potentially help OSS communities to manage their expectations about how many new contributors they should retain by participating in CCEs.

Approach. For obtaining the information on the students, we have used the method depicted in Fig. 1. We have used the term **permanence** to refer to the time in days that a student kept contributing to the assigned project after the end of the CCE.

To determine a CCE's retention rate, it has been necessary to distinguish former CCE's students and project members from newcomers. For distinguishing former CCE's students, we have counted how many participations the students had for that CCE. For distinguishing project members, we have needed to find a suitable timeframe (*threshold*) after which we could consider that a specific contributor refers to a project member (and not a newcomer who started contributing early for being accepted in the CCE). Thus, for each CCE and all students in our sample, we have created a statistical distribution of the intervals between the commits performed before the start of the CCE to arrive at an estimation of this *threshold*. If the interval between the CCE official start date and the date of the commit was higher than the *threshold*, we have considered that the student was a project member. For GSoC, we selected 127 days as the *threshold*, the 99th percentile of the distribution. Thus, any applicant with commits older than 127 days before the GSoC's kickoff was not considered a newcomer. We refer to former CCEs students and contributors with commits older than the *threshold* as **veterans**. **Newcomers** are the first-time CCE participants with commits younger than the *threshold*. A similar approach was used by Colazo and Fang [15], even though they used this method to define **retention**. Here, **retention** has the same meaning as **permanence**. We have used the term **prior permanence** to refer to the period that the students remained before kickoff.

Preliminary results. To understand how long GSoC participants kept committing to their assigned projects after CCE, Fig. 4 depicts the distribution of permanence before and after kickoff. We split our analysis into newcomers and veterans. Fig. 4 (a) and Fig. 4 (c) depict newcomers' permanence, in days, before and after GSoC, while Fig. 4 (b) and Fig. 4 (d) depict the same information but for veterans.

As previously, we only show the students who kept contributing to the project for better data visualization, reporting how many students were removed after the figures' captions, in brackets. Fig. 4 (a) complements a previous finding, by informing that ~33% of newcomers started contributing to their GSoC projects ~90 days before the programs' kickoff (*i.e.*, before knowing they would be accepted).



(a) Newcomers' permanence before GSoC [*128/191 (~67%) did not contribute before*]

(b) Veterans' permanence before GSoC [*38/69 (~55%) did not contribute before*]

(c) Newcomers' permanence after GSoC [*125/191 (~65%) did not remain*]

(d) Veterans' permanence after GSoC [*42/69 (~61%) did not remain*]

Fig. 4. Permanence before and after (retention) distribution for newcomers and veterans (in days)

Fig. 4 (b), shows that veterans were mostly consisted of GSoC former students (56), and the prior permanence refer to previous editions. Fig. 4 (c) shows that the newcomers did not typically keep committing to their projects (~65%), but for the ~35% of the students who did, their permanence on the project lasted around 150 days after GSoC. So, some OSS projects benefited from the newcomers' contributions even after the official program timeline.

In Fig. 4 (d), as with newcomers, we can see that most veterans did not keep committing. The long permanence of the ones who did refers mostly to participation in subsequent GSoC editions, which we consider a different, but valid, type of retention.

Thus, GSoC had a retention rate of 35% for newcomers, who typically contributed additional 150 days to their GSoC projects. For veterans, GSoC had a retention rate of 40%, who typically remained up to 700 days, including the participation in subsequent GSoC editions.

RQ3. What motivates students to join CCEs?

Motivation. As previously mentioned, CCEs may potentially influence the students' decision to voluntarily contribute to OSS, by providing many reward types. Many of these rewards have been linked to developers' retention (see Beecham *et al*. [24]). Students comprise a different population than developers, and mapping the rewards that motivate students to join CCEs can potentially have practical implications for the stakeholders: OSS communities can take advantage of

CCEs by, for instance, attracting newcomers before the engagements; and CCE organizers can, for instance, provide participation certificates.

Approach. To obtain the students' contact information, we used the method depicted in Fig. 1. We designed the study to answer this RQ in two steps. In the first, we have conducted surveys with students[8] and mentors[9] of different CCEs. The surveys were designed following Fink's advice [25]. Questionnaires have been used as they allow for the data collection of a sizeable population and provide relatively standardized data. The questionnaires included open-ended and closed-ended questions.

In the second step, we have conducted semi-structured interviews with some students, mentors, and CCE organizers for validating/deepening the answers. We have crafted the interviews' script[10] following the six types of questions described by Merriam [26]. For analyzing the closed-ended questions, we have been using descriptive statistics. For analyzing the open-ended questions and the interviews, we have been using open coding and axial coding [27]. During open coding, concepts are identified and their properties are discovered in the data. During axial coding, connections between the codes are identified and grouped according to their properties to represent categories [12], [28].

Preliminary results. We conducted a survey with 141 students and 53 mentors who participated in GSoC 2010-2015. Among other things, we asked why did the GSoC students become interested in the program. Table 1 presents the categories yielded from the grouping of the codes and their count (for the GSoC students).

Table 1. Why did you [GSoC student] become interested in the program?

Categories	Count
Money for financial profit	52
Learning experience	42
OSS project gateway	39
Career concerns	29
Work type rewards	30
Money for funding	22
Summer job/project	19
Networking	14
Academic rewards	7
Peer-recognition	3

[8] The GSoC students' questionnaire: https://goo.gl/forms/aEO1DDYqT76dsHED2

[9] The GSoC mentors' questionnaire: https://goo.gl/forms/nHSnfYzVbBgDGkXp2

[10] The interviews' script can be accessed at: https://goo.gl/4El62E

RQ4. What barriers do OSS communities face when trying to incorporate the code produced in CCEs to the projects' codebase?

Motivation. To our best knowledge, no empirical study has tried to understand what are the drawbacks that the OSS communities face when they try to incorporate the students' code into the codebase. We believe that the identification of these barriers can potentially can better equip the OSS communities to take advantage of CCEs.

Approach. We plan to conduct surveys and semi-structured interviews with mentors and other community members to understand the range of tactics that they adopt to incorporate the students' code into the codebase.

RQ5. What factors influence the CCEs students' retention?

Motivation. The early identification of long-term contributors among CCE applicants can potentially help the OSS communities to remain sustainable.

Approach. Our first step is to identify the factors that may influence the students' retention. For this, we plan to review the current literature to obtain a comprehensive set of factors. For instance, previous studies found that contributors' retention (in general) was strongly correlated with prior permanence in the project [6], OSS ideology [17], contribution complexity [29].

Next, we intend to perform qualitative and quantitative analysis of various OSS projects. For the qualitative analysis, we plan to conduct surveys and interviews with students and project members to better understand the relation between the students' willingness to contribute and the factors (context) that lead them to keep/stop contributing. For the quantitative analysis, we plan to use historical data (by mining software repositories) for an objective view of the students' interaction with their projects. We believe that the results of this research can potentially be compiled into a model, which may assist the OSS communities to select the applicants with higher probabilities of becoming long-term contributors.

4 Conclusion

In this research, we have investigated whether the participation in CCEs can lead to better outcomes in terms of more code contributions and new contributors.

We expect that we can provide the OSS communities with practical and valuable information on how to engage more contributors in OSS and also on how to take advantage of CCEs for the benefit of the community.

References

[1] Y. Fang and D. Neufeld, “Understanding Sustained Participation in Open Source Software Projects,” *J. Manag. Inf. Syst.*, vol. 25, no. 4, pp. 9–50, 2009.

[2] R. Farzan, L. Dabbish, R. E. Kraut, and T. Postmes, “Increasing Commitment to Online Communities by Designing for Social Presence,” *Proc. ACM 2011 Conf. Comput. Support. Coop. Work - CSCW '11*, no. March 1923, p. 321, 2011.

[3] E. H. Trainer, A. Kalyanasundaram, C. Chaihirunkarn, and J. D. Herbsleb, “How to Hackathon: Socio-technical Tradeoffs in Brief, Intensive Collocation,” in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing - CSCW '16*, 2016, pp. 1116–1128.

[4] E. H. Trainer, C. Chaihirunkarn, A. Kalyanasundaram, and J. D. Herbsleb, “Community code engagements: Summer of Code & hackathons for community building in scientific software,” in *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, 2014, pp. 111–121.

[5] E. H. Trainer, C. Chaihirunkarn, and J. D. Herbsleb, “The Big Effects of Short-term Efforts: Mentorship and Code Integration in Open Source Scientific Software,” *J. Open Res. Softw.*, vol. 2, no. 1, p. Art. e18, 2014.

[6] A. Schilling, S. Laumer, and T. Weitzel, “Who will remain? - An evaluation of actual Person-Job and Person-Team fit to predict developer retention in FLOSS projects,” *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pp. 3446–3455, 2011.

[7] K. Crowston, N. Jullien, and F. Ortega, “Sustainability of Open Collaborative Communities : Analyzing Recruitment Efficiency Sustainability of Open Collaborative Communities : Analyzing Recruitment Efficiency,” *Technol. Innov. Manag. Rev.*, no. January, pp. 20–26, 2013.

[8] K. R. Lakhani and R. G. Wolf, “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects,” in *Perspectives on Free and Open Source Software*, Cambridge: MIT Press, 2005.

[9] A. Hars and S. Shaosong Ou, “Working for free? Motivations of participating in open source projects,” in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001, vol. 7, p. 9.

[10] J. a. Roberts, I.-H. Hann, and S. a. Slaughter, “Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects,” *Manage. Sci.*, vol. 52, no. 7, pp. 984–999, 2006.

[11] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, “Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects,” *Proc. 18th ACM Conf. Comput. Support. Coop. Work Soc. Comput. - CSCW '15*, pp. 1379–1392, 2015.

[12] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, and D. Redmiles, “The hard life of open source software project newcomers,” *Proc. 7th Int. Work. Coop. Hum. Asp. Softw. Eng. - CHASE 2014*, pp. 72–78, 2014.

[13] F. Fagerholm, A. S. Guinea, J. Münch, and J. Borenstein, “The role of mentoring and project characteristics for onboarding in open source software projects,” *ESEM conf.*, pp. 1–10, 2014.

[14] P. Meirelles, C. Santos, J. Miranda, F. Kon, A. Terceiro, and C. Chavez, “A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects.”

[15] J. Colazo and Y. Fang, “Impact of license choice on open source software development activity,”

J. Am. Soc. Inf. Sci. Technol., 2009.

[16] C. Santos, G. Kuk, F. Kon, and J. Pearson, "The attraction of contributors in free and open source software projects," *J. Strateg. Inf. Syst.*, vol. 22, no. 1, pp. 26–45, 2013.

[17] K. J. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *MIS Q.*, vol. 30, no. 2, pp. 291–314, 2006.

[18] S. Krishnamurthy, S. Ou, and A. K. Tripathi, "Acceptance of monetary rewards in open source software development," *Res. Policy*, vol. 43, no. 4, pp. 632–644, 2014.

[19] I.-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Economic Incentives for Participating Open Source Software Projects," *Twenty-Third Int. Conf. Inf. Syst.*, vol. ICIS 2002, 2002.

[20] J. Tirole and J. Lerner, "Some Simple Economics of Open Source," *J. Ind. Econ.*, vol. 50, no. 2, pp. 197–234, 2002.

[21] J. Lave and E. Wenger, *Situated learning: Legitimate Peripheral Participation*. Cambridge University Press, 1991.

[22] I. S. Wiese, J. T. da Silva, I. Steinmacher, C. Treude, and M. A. Gerosa, "Who is Who in the Mailing List? Comparing Six Disambiguation Heuristics to Identify Multiple Addresses of a Participant," *2016 IEEE Int. Conf. Softw. Maint. Evol.*, pp. 345–355, 2016.

[23] E. H. Trainer, C. Chaihirunkarn, A. Kalyanasundaram, and J. D. Herbsleb, "Community code engagements: Summer of Code & hackathons for community building in scientific software," *Proc. Int. ACM Siggr. Conf. Support. Gr. Work*, pp. 111–121, 2014.

[24] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in Software Engineering: A systematic literature review," *Inf. Softw. Technol.*, vol. 50, no. 9–10, pp. 860–878, 2008.

[25] A. G. Fink, *How to Ask Survey Questions*, Vol 2. SAGE Publications, Inc, 1995.

[26] S. B. Merriam, *Qualitative Research: A Guide to Design and Implementation*, vol. 1. Jossey-Bass, 2009.

[27] A. Strauss and J. M. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. 1998.

[28] I. Steinmacher, M. A. Gerosa, and D. F. Redmiles, "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects," *Proc. ACM Conf. Comput. Coop. Work Soc. Comput.*, pp. 1379–1392, 2015.

[29] S. Dejean and N. Jullien, "Big from the beginning: Assessing online contributors' behavior by their first contribution," *Res. Policy*, vol. 44, no. 6, pp. 1226–1239, 2015.

ON OSS FOUNDATION COMMUNITY SERVICES

Remo Eckert

University of Bern, Switzerland
remo.eckert@iwi.unibe.ch

Abstract. Open source software (OSS) communities are forms of inter-organizational collaborative ecosystems. In case an OSS community is successful and the project grows, there is a need for institutionalization to involve outside parties. The OSS community face a choice of how to organize the community. Either they continue as is, join an existing foundation or they found their own foundation. In this tentative paper, we are going to show that a major motivation why communities join a foundation is due to the related services the foundation provides to them. We structure the provided services in regard to our organizational framework. While some foundations provide services across all three dimensions, other foundations set a clear focus on financial services.

Keywords: Open source software, collaborative software development, organizational affiliation, OSS foundation.

1 Background Of The Research

With respect to the development of OSS, governing the community plays a central role. Governance within OSS communities has been a widely discussed topic for many years [1–3]. By creating a three phase model, de Laat [4] describes the structural evolution of an OSS project. In phase one, governance is spontaneous and explicit coordination and control are non-existent. Phase two introduces internal governance with formal tools, e.g. division of roles, training, modularization or decision-making. This enables an OSS community to be governed internally in order to increase efficiency and effectiveness as the community grows. Eventually, in phase three, in case the OSS community is successful and both companies and other organizations are willing to participate, there is a need for institutionalization to involve outside parties [4]. As OSS communities mature and become economically relevant, those communities face a choice: Either continue as is, create an own OSS foundation or to affiliate with an existing OSS foundation [5].

There are different OSS foundations which offer services to OSS communities as their affiliates. As an example, the Eclipse Foundation has “working groups” as affiliated OSS communities. These working groups include geospatial technologies, Internet of Things or embedded Java systems for the aircraft industry. Similarly, the organization

behind the Linux kernel development, the Linux Foundation, offers a platform for industry-driven OSS development [6]. Their "collaborative projects" currently cover horizontal OSS solutions (widely used in different industries) such as a cloud-computing platform or an embedded Linux platform, as well as vertical OSS solutions (used in a particular industry) such as financial services middleware or a drone open source platform. These practical examples show how mature OSS organizations like the Eclipse Foundation and the Linux Foundation are concerned with more than the initial OSS project they were created for, the Integrated Development Environment respectively the Linux kernel. They offer community management activities to emerging OSS projects as professional services. The foundations share their longstanding experience in community governance and thereby fostering collaborative software development.

While Riehle and Berschneider [5] present an analysis of the structure and processes of OSS developer foundations and distill a model of the structure and governance of foundations, this paper focus on what services an umbrella organization such as the Eclipse Foundation and the Linux Foundation provide for their affiliated communities.

2 Research Motivation & Research Question

The way in which even successful independent OSS projects such as the Node.js JavaScript community joined the Linux Foundation indicates the unpreceded attractiveness of an OSS umbrella organizations [7]. However, it is not clear why OSS communities join an existing OSS foundation. In our work we show that a major motivation why communities join a foundation is due to the related services the foundation provides to them. This research in progress tries to answer the following research question:

RQ: What are the services OSS foundations provide for their affiliated communities?

To answer this question, we developed an organizational framework that consists of different elements of an OSS community. Such a framework helps to understand the needs an OSS community has in order to govern itself. The remainder of this paper is structured as follows: Section 3 illustrates our organizational framework and the meaning of each underlying element. Section 4 we present our research design and section 5 shows the services OSS foundations provide to its affiliated communities. Finally, in section 6 we discuss further research.

3 Organizational Framework Of An OSS Community

With respect to the development of OSS, its community plays a central role. Communities are commonly defined as a group of people who share common values or interests. The common interest of an OSS community is to develop software that is distributed under an OSS license. By looking at the rather general definition of a community it is somewhat related to the term of an organization.

An organization is defined as an entity comprising multiple people that has a collective goal. According to Luhmann [9], there are three characteristics that highlight the organization: First, an organization can decide which people are part of it and which are not. Moreover, the organization can define restrictions and rules and its members have to adhere to the rules, otherwise they can be excluded. Second, organizations have certain goals they want to achieve. Hence, decisions are oriented on these goals. Lastly, organizations exhibit certain hierarchies that regulate the position of its members within the organization.

Besides common goals, roles, rules and structures, most if not all organizations are in need of assets. Assets can either be tangible or intangible and can be owned or controlled to produce a positive economic value. Moreover, they can be converted into cash [10]. From an accounting viewpoint, an asset is a resource controlled by an entity as a result of past events and from which future economic benefits are expected to flow to the entity[1].

The similarities between an organization and OSS communities can be structured in areas derived from organizational theory. Following this, the dimensions of an OSS community are structured in people, organization and assets. In the following subsections, we explain how each of the corresponding elements of people, organization and assets can be understood.

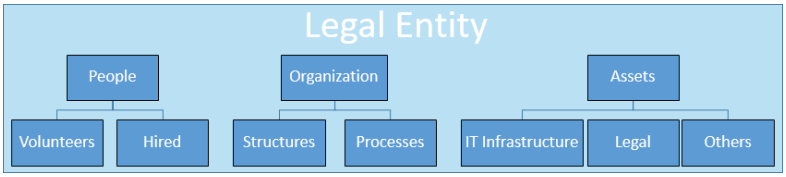


Fig. 1. Different elements of an OSS community.

3.1 People

OSS communities consist of people who virtually find together to share a common goal [11]. The motivation why to contribute to an OSS project differs. Contributors to an OSS project can either by paid by an employer or they can be volunteers. Generally, motivation can be distinguished in intrinsic and extrinsic motivation. An action is extrinsically motivated when it is performed to obtain some separable outcome whereas an intrinsically motivated action is done for the mere interest or joy of performing it [12]. However, the motivations to contribute to an OSS projects of hired people and volunteers differ [13]. A developer's "itch worth scratching", as stated by Raymond, might be not as strong for a paid developer as for a volunteer [14]. Tasks such as the project design, coordination, testing, documentation and bug fixing are less attractive

[1] http://www.ifrs.org/

for volunteers and could therefore be done by hired people to ensure that these tasks are done [15]. We therefore distinguish between hired and volunteers.

3.2 Organization

Some OSS communities, especially bigger ones, have formal membership rules and agreements such as the membership fee and bylaws with different roles and functions [16]. Bylaws are rules established by the community to regulate itself in a structural and a processual way. As an example, the Eclipse Foundation bylaws regulate the overall purpose of the community, the powers and duties of the different roles within the community, how and when the members are elected and how meetings are organized. Moreover, the Eclipse Foundation bylaws regulate how decisions are made, explain the tasks of the different committees, councils as well as the different boards and the different forms of membership [17]. Contributions not only evolve the software but also redefine the role of the contributors and thus changes the social dynamics of the community. Consequently, project leaders and core members should not only focus on the evolution of the software, but also on creating an environment and culture which foster and encourage new members to move toward the center of the OSS community through continual contributions by informal as well as formal mechanisms. Such mechanisms allow developers to work independently by encouraging or reinforce other developers to work in ways that are expected by the community [18]. We therefore distinguish between structures and processes.

3.3 Assets

If an OSS wants to get contributions, it needs to market itself, across the market or within the community. This includes hosting a website with the published source code of the OSS project. To do so, an IT infrastructure is required. For example, in order to push changes, a committer needs access to a copy of the latest version of the project's source code. This is usually done by relying on decentralized technologies, such as a distributed repository and some form of version control. Normally, this is done using a decentralized version control system (DVCS) such as GIT where collaboration among various developers is possible [19]. However, a DVCS and websites need to run on a server where access via Internet is guaranteed. As described by German [15], at the level of community infrastructure, servers as well as bandwidth are required to communicate and share progress of the collaborators. Although OSS do not fully accomplish conditions to be included in financial reports as an asset [20], it can be protected in different ways such as trademarks and brands. A common practice of OSS foundations is to own the copyright of the source code and related texts, as stated by Riehle [21]. Having a single and central copyright holder means that it holds the asset and therefore can protect it easier compared to the situation when hundreds of contributors holding individual rights of their parts. Further, a legal entity ensure to protect volunteer contributors from individual liability, enter into agreements collectively and protect their code, trademarks, licenses and brands [22]. Because most of the work in an OSS community is done by globally distributed individuals, face to face meetings are rare.

However, they help to better communicate and resolve potential conflicts. Consequently, an infrastructure such as rooms for meetings and an internet connection are needed. We therefore distinguish between three different assets: IT Infrastructure, legal assets and other assets.

4 Research Design

This research in progress shows the result of an exploratory theory generation process for the affiliation of OSS communities to an existing OSS foundation. To answer the research question, we are going to show services an OSS foundation provide and structure them in regard to our organizational framework. To get intersubjectivity, the author of this paper and a second researcher did the research independent and discussed and merged the different results in a table.

4.1 Data Collection And Analysis

Our sample consists of 7 OSS foundations which have affiliated communities. While some OSS foundations offer a wide range of different services (e.g. the Eclipse Foundation), others are more lightweight and only provide services in a given area (e.g. the Software in the Public Interest acts as a fiscal sponsor). Unlike random sampling, we chose the approach of maximum variation cases, therefore our analyzed OSS foundations show distinct characteristics and processes [8]. Our 7 OSS foundations are: *the Eclipse Foundation (EF), the Linux Foundation (LF), the Apache Software Foundation (ASF), the Software in the Public Interest (SPI), the Software Freedom Conservancy (SFC), the OW2 Association (OW2)* and *the OS Geospatial Foundation (OSG)*. Our analysis is based on publicly accessible documents from the foundations or its affiliated communities.

5 Services of an OSS Foundation

The following section will give a description which services a foundation can provide for its affiliated communities, based on the provided services of our 7 chosen OSS foundations.

Hired: As an established foundation already has employed staff, it can provide its communities with shared labor. The staff can reach from a shared community manager helping to establish and maintain a healthy community, to legal consultants helping in the intellectual property rights (IPR) management, back-office work such as membership-collection and accounting and various others. The foundation can provide already existing services such as recruiting, payroll-services, insurance-handling and others. As an example, the SFC provides financial services such as the funding of the communities. To do so, it employs full time employees which are paid by the different affiliated projects who need to pay a certain amount of their revenue that SFC processes.

Volunteers: Ecosystem Marketing & Development: As a "community-forming" service, a foundation can substantially raise the attention within an already existing ecosystem. Because an established foundation is already part within an ecosystem, the foundation can help its affiliated communities to get new members. As an example, a foundation can organize events, conferences, marketing events or it can provide online resources (such as social media marketing) to get attention within the OSS ecosystem. Such occasions can provide a community with the opportunity to meet developers from the already existing OSS ecosystem. This strengthens the community because affiliated communities can share knowledge among each other while promoting their own projects. Furthermore, events and conferences can be used to enlarge the community of the affiliated organization and to boost interest in the OSS community itself. As an example, OW2 organizes tradeshows and conferences such as the annual OW2 conference where affiliated project members and other interested people can meet.

Structures: With their experience and given legal entity, a foundation can provide its communities a structural framework for reuse. Affiliated communities can profit from established bylaws with structural elements of different boards and committees, policies on membership dues, license policies or IPR policy. Contracts and legal documents are ready for reuse. The given bylaws can regulate the purpose of the community, power and duties of the different roles as well as other organizational elements such as how decisions are made. Moreover, a foundation can help to establish a business plan and a vision for its communities. As an example, the Eclipse Foundation provides its affiliated communities with bylaws which are ready to use.

Processes: Affiliated communities can benefit from a written software development process of the foundation. As an example, the Eclipse Foundation has a formal development process for large-scale and distributed development which can be adopted by its affiliated communities[2]. In its core, the development process relies on openness, transparency and meritocracy. **IPR-management:** The co-developed software needs to be managed to make sure it is compliant and does not infringe any legal rights of others. An established foundation, such as the Eclipse foundation, can provide its affiliated communities with an IPR process to help that all contributions made don't infringe rights of others. To do so, every contribution has an own copyright holder and the contribution is made publicly available under the Eclipse public license. Therefore, all contributors are required to sign a contributor license agreement where they accept that their original work is being contributed under the EPL. In a second step, the Eclipse foundation analyzes all contributions. In this process, contributions are analyzed if their license is compatible with the EPL. The end result is a software product which can safely be distributed in commercial products. **Financial services:** The foundation can help to handle the membership application-process, membership-renewals, manage the

[2] https://eclipse.org/projects/dev_process/development_process.php

billing process for membership fees and the related back-office work such as accounting. As an example, the SPI handles the donation process for its affiliated communities. Donations can be made to the SPI to a specific affiliated project or to the SPI in general. The affiliated community can then request the payment of project expenses such as infrastructure or other community expenses. The remaining money is held in trust by the SPI for the community and the SPI is doing all the related back-office work.

IT Infrastructure: A foundation with its existing IT infrastructure can provide different services related to the IT infrastructure. It can host entire projects on their infrastructure including code-repositories, mailing lists, issue trackers, newsgroups, wikis, download-sites as well as websites. Providing those services, the foundation enables not only the development process as well as internal communication between the communities but it also helps the community to market itself by hosting a website where the community can promote itself and provide access to the software. As an example, OSGeo provides its affiliated communities with source code control systems, issue trackers, mailing lists and web hostings.

Legal: **Intellectual property management:** Although OSS does not fully accomplish conditions to be included in financial reports as an asset [23], it can be protected in different ways such as with trademarks and brands. The foundation can help to protect assets of their affiliated communities. Affiliated communities may transfer trademarks and brands to the umbrella organization. As an example, the SFC can hold any assets for an affiliated project: copyrights, trademarks, domain names or computer equipment. Further, the foundation can allow its affiliated communities to brand their project under their name. Doing so, affiliated communities may profit from an existing reputation of the umbrella organization **Shelter for legal suits:** Another way an affiliated community may profit is by running under the legal umbrella of the foundation. Doing so, the community is able to use a sustainable framework to operate within. Moreover, a foundation can provide protection against individual liability because in a lawsuit, only the legal entities assets and not those from individuals are potentially liable. As an example, the ASF allow individual volunteers to be sheltered from legal suits directed at the foundations projects and individuals benefit of protection from personal liability of their work on the project.

Others: Legal organizations need a physical address and the foundation can provide the same address to their affiliated communities. Moreover, the foundation can give access to their premises. A community may then attend meetings in a room at the foundations location or at conferences organized by the foundation as it is the case on some events organized by the Linux Foundation.

Table 1 shows the results of our data analysis, the references can be found in the appendix.

Area	Dimension	Service	Potential community-benefit	LF	EF	ASF	SPI	SFC	OW2	OSG
People	Hired	Shared employees	Cost sharing, know-how sharing	X^{1}	X^{2}	X^{3}	X^{4}	X^{5}	X^{6}	X^{7}
	Volunteers	Ecosystem Marketing	Getting new contributors	X^{8}	X^{9}	X^{10}		X^{11}	X^{12}	X^{13}
Organization	Structures	Help in establishing a structure (e.g. by-laws)	Shorter time-to-market	X^{14}	X^{15}	X^{16}		X^{17}	X^{18}	X^{19}
	Processes	Software Development Guidance	Preventing potential conflicts	X^{20}	X^{21}	X^{22}			X^{23}	X^{24}
		Financial Services	Reducing administration overhead	X^{25}	X^{26}	X^{27}	X^{28}	X^{29}		X^{30}
		Legal support	Preventing potential conflicts	X^{31}	X^{32}	X^{33}	X^{34}	X^{35}	X^{36}	X^{37}
		IPR management, IP Policy	Preventing potential conflicts	X^{38}	X^{39}	X^{40}	X^{41}	X^{42}	X^{43}	
Assets	IT Infrastructure	IT Infrastructure management	Reducing administration overhead	X^{44}	X^{45}	X^{46}		X^{47}	X^{48}	X^{49}
	Legal	IP Management (Holding Trademarks & Brands)	Reducing administration overhead	X^{50}	X^{51}	X^{52}	X^{53}	X^{54}	X^{55}	X^{56}
		Shelter for legal suits	Preventing potential conflicts	X^{57}	X^{58}	X^{59}	X^{60}	X^{61}		
	Others	Shared rooms	Rooms at conferences	X^{62}	X^{63}	X^{64}		X^{65}	X^{66}	X^{67}

Table 1. Services of 7 OSS foundations.

6 Discussion And Further Research

Our research in progress shows the different services OSS foundations provide for their affiliated projects. The services can be structured according to our organizational framework in the dimensions people, organization and assets. While some foundations provide services across all three dimensions, other foundations set a clear focus on financial services. As an example, the Software in the Public Interest mainly accept donations and hold funds and assets on behalf of associated projects. Certain foundations such as the OSGeo Foundation are specialized for communities in a specific area, in the case of OSGeo in the area of geospatial technology.

OSS communities benefit from the provided services of the foundation, however, certain restrictions may emerge. Communities need to balance the provided services with the restrictions and cost. The Eclipse Foundation, as an example, provides affiliated communities with bylaws which cannot be changed by the affiliated communities and need to be accepted to join as an affiliate project. Moreover, some OSS foundations require a specific license the software needs to have in order to join as an affiliate project or require to transfer IPR to the foundation.

The needs may differ from community to community. Further research could distinguish the different needs OSS communities have and how the foundation and its affiliated communities interact with each other. While the provided services offer communities various benefits, a community may decide to make its own foundation. Further research could try to shed some light into the advantages and disadvantages of being affiliated to an umbrella organization or being autonomous.

7 References

1. De Noni, I., Ganzaroli, A., Orsi, L.: The evolution of OSS governance: a dimensional comparative analysis. Scandinavian Journal of Management. 29, 247–263 (2013).
2. Franck, E., Jungwirth, C.: Reconciling rent-seekers and donators–The governance structure of open source. Journal of Management and Governance. 7, 401–421 (2003).
3. Schaarschmidt, M., Walsh, G., von Kortzfleisch, H.F.O.: How do firms influence open source software communities? A framework and empirical analysis of different governance modes. Information and Organization. 25, 99–114 (2015).
4. de Laat, P.B.: Governance of open source software: state of the art. Journal of Management & Governance. 11, 165–177 (2007).
5. Riehle, D., Berschneider, S.: A Model of Open Source Developer Foundations. In: Open Source Systems: Long-Term Sustainability. pp. 15–28. Springer Berlin Heidelberg (2012).
6. The Linux Foundation: Becoming a Linux Foundation Collaborative Project: Working Together to Help Projects Grow, http://collabprojects.linuxfoundation.org/sites/collabprojects/files/lf_collaborative_projects_brochure.pdf. (2014).
7. The Linux Foundation: Node.js Foundation Advances Community Collaboration, Announces New Members and Ratified Technical Governance, http://www.linuxfoundation.org/news-media/announcements/2015/06/nodejs-foundation-advances-community-collaboration-announces-new. (2015).
8. Flyvbjerg, B.: Five Misunderstandings About Case-Study Research. Qualitative Inquiry. 12, 219–245 (2006).
9. Luhmann, N.: Funktionen und Folgen formaler Organisation. Berlin: Duncker & Humblot (1964).
10. O'Sullivan, A., Sheffrin, S.M.: Economics: Principles in Action. Upper Saddle River, New Jersey: Pearson Prentice Hall. (2003).
11. Rheingold, H.: The Virtual Community: Homesteading on the Electronic Frontier. Cambridge, Mass: MIT Press (2000).
12. Deci, E.L., Ryan, R.M.: The general causality orientations scale: Self-determination in personality. Journal of Research in Personality. 19, 109–134 (1985).
13. Roberson, Q.M., Stewart, M.M.: Understanding the motivational effects of procedural and informational justice in feedback processes. British Journal of Psychology. 97, 281–298 (2006).
14. Raymond, E.S.: The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media, Inc., Sebastopol, CA, USA (2001).
15. German, D.M.: The GNOME project: a case study of open source, global software development. Software Process: Improvement and Practice. 8, 201–215 (2003).
16. von Krogh, G., Spaeth, S., Lakhani, K.R.: Community, joining, and specialization in open source software innovation: a case study. Research Policy. 32, 1217–1241 (2003).
17. The Eclipse Foundation: BYLAWS OF ECLIPSE FOUNDATION, INC. https://eclipse.org/org/documents/Eclipse%20BYLAWS%202003_11_10%20Final.pdf,

https://eclipse.org/org/documents/Eclipse%20BYLAWS%202003_11_10%20Final.pdf, (2003).
18. Jensen, C., Scacchi, W.: Collaboration, leadership, control, and conflict negotiation and the netbeans.org open source software development community. In: System Sciences, 2005. p. 196b–196b. IEEE (2005).
19. Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., German, D.M.: Open source-style collaborative development practices in commercial projects using github. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1. pp. 574–585. IEEE Press (2015).
20. García-García, J., Alonso de Magdaleno, M.I.: VALUATION OF OPEN SOURCE SOFTWARE: HOW DO YOU PUT A VALUE ON FREE?, (2013).
21. Riehle, D.: The Economic Case for Open Source Foundations. Computer. 43, 86–90 (2010).
22. O'Mahony, S.: Guarding the commons: how community managed software projects protect their work. Research Policy. 32, 1179–1198 (2003).
23. García-García, J., Alonso de Magdaleno, M.I.: VALUATION OF OPEN SOURCE SOFTWARE: HOW DO YOU PUT A VALUE ON FREE? Revista de Gestão, Finanças e Contabilidade 3.1. (2013).

8 Appendix

X^1	https://www.linuxfoundation.org/projects/services/public-relations-social-media-video
X^2	https://eclipse.org/org/foundation/reports/annual_report.php & https://dev.eclipse.org/mhonarc/lists/ide-dev/msg01077.html
X^3	https://www.apache.org/foundation/how-it-works.html & https://www.apache.org/foundation/governance/orgchart
X^4	http://www.spi-inc.org/donations/
X^5	https://sfconservancy.org/projects/services/ & https://sfconservancy.org/projects/apply/
X^6	https://www.ow2.org/bin/download/Membership_Joining/Legal_Resources/OW2-2015-ReportOnFinanceAndOperations.pdf
X^7	http://www.osgeo.org/content/faq/foundation_faq.html
X^8	https://www.linuxfoundation.org/projects/services/community-and-ecosystem-enablement & https://www.linuxfoundation.org/projects/services/creative-services & https://www.linuxfoundation.org/projects/services/public-relations-social-media-video
X^9	http://www.eclipse.org/org/workinggroups/about.php#wg-neutral
X^{10}	https://www.apache.org/foundation/governance/
X^{11}	https://sfconservancy.org/projects/services/ & https://sfconservancy.org/about/
X^{12}	https://www.ow2.org/bin/view/About/OW2_Consortium#OW2_business_ecosystem
X^{13}	http://www.osgeo.org/content/foundation/about.html & http://www.osgeo.org/content/faq/foundation_faq.html

X^{14}	https://www.linuxfoundation.org/projects/services/governance-and-intellectual-property & https://www.linuxfoundation.org/projects/services/project-setup-and-launch
X^{15}	https://eclipse.org/org/documents/Eclipse%20BYLAWS%202011_08_15%20Final.pdf
X^{16}	https://www.apache.org/foundation/how-it-works.html
X^{17}	https://sfconservancy.org/projects/services/
X^{18}	https://www.ow2.org/bin/view/Collaborative_Projects/
X^{19}	http://www.osgeo.org/content/foundation/about.html
X^{20}	https://www.linuxfoundation.org/projects/services/open-source-compliance
X^{21}	https://eclipse.org/projects/dev_process/development_process.php
X^{22}	http://www.apache.org/legal/release-policy.html
X^{23}	https://projects.ow2.org/bin/view/wiki/oscar
X^{24}	http://www.osgeo.org/incubator/process/codereview.html
X^{25}	https://www.linuxfoundation.org/projects/services/finance-operations-human-resources
X^{26}	http://www.eclipse.org/org/workinggroups/polarsys_charter.php
X^{27}	https://www.apache.org/foundation/sponsorship
X^{28}	http://www.spi-inc.org/donations/
X^{29}	https://sfconservancy.org/projects/services/
X^{30}	http://www.osgeo.org/content/foundation/about.html
X^{31}	https://www.linuxfoundation.org/open-source-professionals
X^{32}	http://www.eclipse.org/org/workinggroups/about.php#wg-ip
X^{33}	https://www.apache.org/foundation/how-it-works.html
X^{34}	http://www.spi-inc.org/projects/services/
X^{35}	https://sfconservancy.org/projects/services/
X^{36}	https://www.ow2.org/bin/view/Membership_Joining/Membership_Benefits
X^{37}	http://www.osgeo.org/content/foundation/about.html
X^{38}	https://www.linuxfoundation.org/projects/services/open-source-compliance
X^{39}	https://eclipse.org/org/documents/Eclipse_IP_Policy.pdf
X^{40}	http://www.apache.org/foundation/marks/
X^{41}	http://www.spi-inc.org/corporate/resolutions/1998/1998-11-16.iwj.2/
X^{42}	https://sfconservancy.org/news/2015/jul/15/ubuntu-ip-policy/
X^{43}	https://www.ow2.org/bin/download/Membership_Joining/Legal_Resources/OW2C-IPR.pdf
X^{44}	https://www.linuxfoundation.org/projects/services/it-infrastructure-management
X^{45}	http://www.eclipse.org/org/workinggroups/about.php#wg-neutral
X^{46}	https://www.apache.org/dev/services.html

X^{47}	https://sfconservancy.org/about/ & https://sfconservancy.org/projects/services/
X^{48}	https://www.ow2.org/bin/view/About/OW2_Consortium & https://www.ow2.org/bin/view/About/FAQ#FAQ.1_OW2_services
X^{49}	http://www.osgeo.org/content/faq/foundation_faq.html & http://www.osgeo.org/content/foundation/about.html
X^{50}	https://www.linuxfoundation.org/projects/services/governance-and-intellectual-property & https://www.linuxfoundation.org/projects/services/open-source-compliance
X^{51}	http://www.eclipse.org/org/workinggroups/about.php#wg-neutral
X^{52}	https://www.apache.org/foundation/how-it-works.html
X^{53}	http://www.spi-inc.org/projects/services/ & http://www.spi-inc.org/corporate/resolutions/2004/2004-08-10.iwj.1/ & http://www.spi-inc.org/corporate/trademarks/
X^{54}	https://sfconservancy.org/projects/services/
X^{55}	https://www.ow2.org/bin/view/About/FAQ#FAQ.IPRs_to_OW2 & https://tc.ow2.org/bin/view/wiki/Intellectual_Property
X^{56}	http://www.osgeo.org/content/faq/foundation_faq.html & http://www.osgeo.org/content/foundation/legal/licenses.html
X^{57}	https://www.linuxfoundation.org/terms
X^{58}	https://eclipse.org/legal/termsofuse.php
X^{59}	https://www.apache.org/foundation/how-it-works.html
X^{60}	http://www.spi-inc.org/projects/services/ & http://www.spi-inc.org/corporate/resolutions/2004/2004-08-10.iwj.1/
X^{61}	https://sfconservancy.org/projects/services/
X^{62}	https://www.linuxfoundation.org/projects/services/event-management
X^{63}	http://www.eclipse.org/org/workinggroups/about.php#wg-neutral
X^{64}	http://events.linuxfoundation.org/events/archive/2016/apachecon-north-america/program/schedule
X^{65}	https://sfconservancy.org/projects/services/
X^{66}	https://www.ow2.org/bin/view/About/FAQ#FAQ.Does_OW2_organise_events
X^{67}	http://www.osgeo.org/conference

Appendix 1. References of the services.

Analysis and Prediction of Log Statement in Open Source Java Projects

Sangeeta Lal[1], Neetu Sardana[1], and Ashish Sureka[2]

[1] Jaypee Institute of Information Technology (JIIT), India
sangeeta@jiit.ac.in,neetu.sardana@jiit.ac.in
[2] ABB Corporate Research, India
ashish.sureka@in.abb.com

Abstract. Log statements present in the source code provide important information about program execution which is helpful in several software development activities such as remote issue resolution, debugging, and load testing. However, log statements have a trade-off between cost and benefit and hence it is important to optimize the number of log statements in the source code. Previous studies show that optimizing log statements in the source code is a non-trivial activity and software developers often face difficulty in it. Several previous studies empirically analyze log statement and propose models for logging prediction. However, there are gaps in the literature which our study aims to address.
In this work, we aim to build tools and techniques which can help developers in optimizing the number of log statements in the source code. In order to do so, we first start by performing a survey of software developers from open-source projects. We then analyze properties of logged and non-logged code constructs at multiple levels. Using inputs from our empirical analysis we then propose machine learning based models for within-project catch-blocks and if-blocks logging prediction. We extend this study for cross-project logging prediction using ensemble based machine learning techniques. Our initial results are encouraging and show the possibility of making a robust machine learning based logging prediction tool for Java projects.

Keywords: Debugging, Logging, Machine Learning, Source Code Analysis, Tracing

1 Problem statement and its importance

Logging is a software development practice that is performed by inserting log statements in the source code. Log statements are used to record execution information about the program. For example, Listing 1.1 shows a try/catch block taken from the Tomcat project. This catch-block consists of a log statement which records the information about *login failure*. This recorded log can be used by the software developers at the time of debugging, in a case of login failure.

Log statements are customizable and provide feature for verbosity level modification. Hence, logging is a better alternative to commonly used *print* statements for debugging. In a survey performed by Fu et al. [4], 96% of the survey respondents agreed that log statements are important in system maintenance and development. In addition to debugging [20], logging is useful several other software development activities such as remote issue resolution [1] and load testing [6, 7].

Listing 1.1: Example of a try/catch-block code snippet from the Tomcat project

```
try{
  lc=new LoginContext(getLoginConfigName());
  lc.login();
}catch(LoginException e)
{
 log.error(sm.getString( spnegoAuthenticator.serviceLoginFail ),e);
 response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
 return false;
}
```

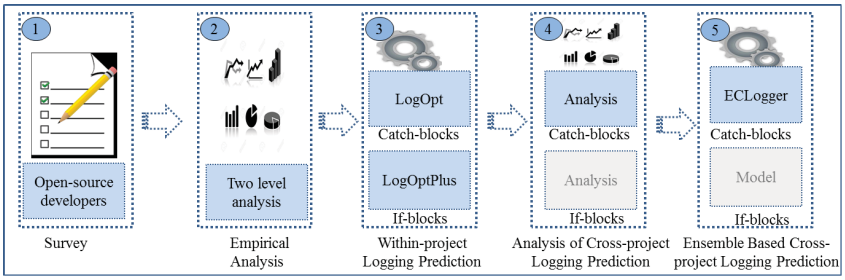


Fig. 1: Illustration of five step study objective to improve logging in open-source Java projects

Log statements are beneficial but they have trade-off between cost and benefit [4, 22]. Logging is an I/O intensive activity and hence excess of log statements in the source code can cause performance overhead to the system. For example, excessive or unnecessary logging is considered as one of the top 5 reasons for performance bottlenecks in .NET applications [5]. In addition, excess of log statements can generate too many trivial logs and can make debugging harder. Similar to excess logging sparse logging is also not good. Sparse logging can miss important debugging information and can potentially make debugging harder. Shang et al. [17] reported a case of a user who was complaining about less logging of catch-blocks in Hadoop project. Hence, it is important to optimize the number of log statements in the source code.

Previous research shows that optimizing log statements in the source code is a non-trivial task and software developers often face difficulty in it [4, 22]. In a survey by Zhu et al. [22], 68% of the participants said that they face logging

difficulties. Optimized source code logging is particularly challenging for OSS because source code logging is often based on domain knowledge & experience of software developers. Contribution to OSS is voluntary and hence, such domain knowledge is seldom documented in the real world. A study by Levesque [13] shows that OSS lack documentation. In addition to this, finding mentors in OSS is a challenging task. Due to which source code becomes challenging for new OSS developers. We believe that automated tools and techniques that can aid OSS developers in logging can be beneficial in improving both OSS product and process. New OSS developers can get guidance about source code logging at the times of coding which will improve the quality of OSS product. Similarly, automated logging checking tool can be run before every commit or release which can generate warnings about code constructs that need to be logged and hence will improve the OSS process.

Several recent studies work on empirically analyzing and predicting log statements in the source code for C# projects [4, 22]. These studies show that logged and non-logged code constructs have differentiating properties for C# projects. For example, Fu et al. [4] reported that catch-blocks consisting of *'FileNotFoundException'* and the corresponding try-block consisting of the keyword *'delete'* are often not logged. However, there does not exists any large scale & focused study which analyze properties of logged and non-logged code constructs for Java projects. Previous studies show that machine learning based models can be beneficial in predicting logged code constructs and propose machine learning based logging prediction models for C# projects [4, 22]. At present, there is no machine learning based model for logging prediction on Java projects. In addition to this, cross-project logging prediction is relatively unexplored yet. Logging prediction models for Java code constructs can be beneficial to the software developers because Java is one of the widely used programing language [2]. Therefore, my dissertation explore following:

"*Analyzing logged and non-logged code constructs to build machine learning based logging prediction model for Java project*".

To address the above, this work is broadly divided into five parts i.e, Research Objectives (RO's), also illustrated in Figure 1:

- **RO 1:** Analysis of open-source developers opinion towards source code logging on Java projects.
- **RO 2:** Perform a large-scale and in-depth study of logged and non-logged code constructs for Java projects.
- **RO 3:** Build a machine learning based within-project logging prediction model for Java projects.
- **RO 4:** Analysis of machine learning based logging prediction models for cross-project logging prediction for Java projects.
- **RO 5:** Build an ensemble of classifier based cross-project logging prediction model for Java projects.

Literature shows that currently there is no in-depth study on logged code constructs analysis or prediction for Java projects (refer to section 2). To address this gap, in RO 1, we first perform a survey of open-source developers to identify their opinion about logging on Java projects. The outcome of the survey reveals that software developers face difficulty in source code logging in Java project. We believe that automated tools and techniques that can aid software developers in source code logging can be beneficial. Hence, we propose machine learning based logging prediction models. The main challenge in machine learning based logging prediction is the identification of good features for training the machine learning model. To identify these features, in RO 2, we perform a large scale, focused and two-level empirical analysis of logged and non-logged Java code constructs. A code construct is consider as logged if it consists of at least one log statement otherwise it is considered as non-logged. This study reveal that there are several distinguishing features of logged and non-logged code constructs. Using some of the findings of our empirical study, in RO 3, we propose machine learning based logging prediction model for catch-blocks and if-blocks logging prediction model. We start by catch-blocks and if-blocks because they are one of the most frequent logged code snippets [14]. Empirical study of these machine learning based models show that they are effective in within-project (i.e., when training and testing data is from the same project) logging prediction. The performance of these models was not tested for cross-project (i.e., when training data and testing data are from the different projects) logging prediction. Hence, in RO 4, we analyze performance of several classifiers for cross-project logging prediction. The analysis reveals that different classifiers are complementary to each other for the task of cross-project catch-blocks logging prediction. Using this finding, in RO 5, we propose an ensemble of classifier based approach for cross-project catch-blocks logging prediction. This thesis provides a comprehensive study of source code logging with respect to Java projects. Following are the main scientific and technical contributions made by this thesis:

Scientific Contribution: In the literature, there are studies on logged and non-logged code constructs only for C\C++\C# projects. This thesis fills this significant research gap and perform in-depth study of logging in Java code constructs. The survey of open source developers indicates a need towards logging prediction tool for Java code constructs. The empirical study reveals the presence of distinguishing properties among logged and non-logged Java code constructs. This study propose machine learning based logging prediction models for catch-blocks and if-blocks logging. Analysis of these models on two Java projects reveals that machine learning based logging prediction models are effective in logged code construct prediction for Java projects. This study performs comprehensive analysis of several machine learning classifiers for cross-project catch-blocks logging prediction and show the effectiveness of using ensemble of classifier based approach for cross-project catch-blocks logging prediction.

Technical Contribution: This thesis have proposed several conceptual models

for logging prediction Java project. ***LogOpt*** [12] and ***LogOptPlus*** [11] models proposed in this thesis are useful for within-project catch-blocks and if-blocks logging prediction for Java projects. ***ECLogger*** [9] models proposed in this thesis is useful in improving the performance of cross-project catch-blocks logging prediction performance of Java projects.

2 Related Work

2.1 Empirical Analysis of Log statements

Fu et al. [4] analyze 100 randomly selected log statements from two closed-source systems (written in C#). They categories the log statements in five categories: assertion-check, return-value-check, exception, execution points, logic-branch and observing-point logging. They further perform detailed study of 70 non-logged catch-blocks and find reasons of not-logging. Several other studies analyze different aspect of logs. Yuan et al. [19] analyze 250 randomly sampled bug reports from five large C/C++ projects to find efficacy of logs for debugging. They report that most of the failures can not be diagnosed using existing logs. In another study, Yuan et al. [20] analyze four open source projects written in C/C++. They investigate amount of effort that software developers spend on modifying existing log statements. Shang et al. [16] analyze logs and report that log statements are often modified without considering the underlying application that use logs, and hence, log modification often results in break of functionality in log processing applications.

As the literature shows, currently there is no large scale & in-depth study of logged and non-logged code constructs for Java projects. Hence, as part of this work, we first perform a survey of software developers from open-source Java projects. In order to know their opinion about logging in Java projects (refer to section 3.1). We then perform, a two level, in-depth, and large scale study of logged and non-logged code constructs for open-source Java projects (refer to section 3.2).

2.2 Logged Code Construct Prediction

Several prior studies focus on improving logging statements in the code. Yuan et al. [21] propose *LogEnhancer* tool to enhance the content of log statements by adding causally-related information to improve failure diagnosis. Enhancing content of log statements is important but it does not consider cases of code snippets which are not logged. Yuan et al. [19] propose *ErrorLog* tool to log all the generic exceptions in C# code. This study shows the first steps towards automated logging but logging all the generic exceptions can cause too many log statements. Fu et al. [4] show the uses of machine learning based model for logging prediction. Zhu et al. [22] extended the study performed by Fu et al. and propose, *LogAdviosr*, an improved machine learning based model for logging prediction on C# projects. While there have been studies on machine

learning based logging prediction for C# projects, there has been no research study which focuses on logging prediction on Java projects. Hence, in this work we propose two machine learning based models for catch-blocks and if-blocks logging prediction for Java projects (refer to section 3.3).

2.3 Cross-project logging prediction

Cross-project logging prediction is important, as in real world there are many new or small projects. These projects do not have the sufficient amount of prior data to train the machine learning model. In such cases, it is important to train the prediction model from other large and long live projects. Cross-project prediction have been explored in detail for other software development activities such as defect prediction [15] and build co-change [18] prediction. However, cross-project logging prediction is relatively unexplored yet. Zhu et al. [22] perform an experiment related to cross-project logging prediction. They report that performance of cross-project logging prediction is degraded considerably as compared to within-project logging prediction. In this work, we perform in-depth analysis of several machine learning algorithms for cross-project catch-blocks logging prediction (refer to section 3.4) and also propose an ensemble of classifier based approach to improve cross-project logging prediction performance (refer to section 3.5).

3 Approach

In the following subsections, we give brief detail about each RO. We discuss motivation, research questions (RQ's), and approach used to answer these RQ's.

3.1 RO 1: Survey of Open-source Software Developers

We conduct a cross-sectional survey of developers from several open-source projects. We create a survey using SurveyMonkey [3] website and create a web link to it. We sent an email with the link to our survey to the project mailing lists (for example: chromium-dev@chromium.org) of Google Chromium, Apache OpenOffice, Apache Tomcat, Geronimo and Presto projects. Our survey consists of five mandatory questions. Table 2 lists all the five survey questions. In this survey, we ask software developers about their opinion towards source code logging. For example, we ask questions related to *'frequency of log churning*, *'log verbosity level assignment'*, and *'what & where to log ?'*. We believe that inputs from practitioners are needed to inform our research on logging.

[3] https://www.surveymonkey.com/

3.2 RO 2: Empirical Analysis of Logged and Non-logged Code Constructs

We perform two level -*high level (file level)* and *low level (catch-block)* level-analysis of logged and non-logged code on Java projects. We perform ***statistical-analysis*** of logged and non-logged files and answer three RQ's. First, we analyze the distribution of logged files in total files. Second, we analyze the complexity of logged and non-logged files. We use Source Lines of Code (SLOC) of a file as a measure of complexity for logged and non-logged files. Third, we analyze co-relation between file complexity and its log statement count.

We perform ***statistical-analysis*** of logged and non-logged catch-blocks and answer five RQ's. First, we analyze complexity of try-blocks associated with logged and non-logged catch-blocks. We measure complexity using three parameters: SLOC, number of arithmetic operators, and number of functions called in try-blocks. Second, we analyze logging ratio of all the exception types. Third, we analyze the contribution of all the exception types in total catch-blocks and in logged catch-blocks. Fourth, we identify top-20 exception types and compare their logging ratios across the projects. Fifth, we analyze whether a single try-block can have both logged and non-logged catch-blocks or not. We use LDA to perform ***content-analysis*** of try-blocks associated with logged and non-logged catch-blocks. We perform content analysis to find whether try-blocks associated with logged and non-logged catch-blocks consists of different topics on not. LDA is a popular topic modeling technique, but currently there is no study which analyzes topics present in try-blocks associated with logged and non-logged catch-blocks.

We perform two level analysis to explore whether logged and non-logged code constructs have differentiating properties at both the levels or not. Multi-level analysis is useful in feature identification phase of logging prediction tools. As it gives insights about which part of the source code is more useful for feature extraction for training the logging prediction model.

3.3 RO 3: With-in Project Logging Prediction

We use findings from our previous work [10] for feature set identification. We extract 46 and 28 distinguishing features for catch-blocks and if-blocks logging prediction. We categories each features based on its *domain*, *type* and *class*. *Domain*, identifies the part of source code from where the feature is extracted. *Type*, identifies whether the feature is boolean, numeric or textual. *Class*, identifies whether the feature is positive or negative. We extract features from different domains, as our previous study show that both high level and low level code construct show differentiating properties towards logged and non-logged code constructs. Using these features we propose two machine learning based models: ***LogOpt*** [12] and ***LogOptPlus*** [11]. *LogOpt* for catch-blocks and *LogOptPlus* for if-blocks logging prediction for Java projects. In this work, we answer two RQ's: first, we analyze properties of all the proposed features, second, we analyze the performance of the proposed machine learning models for the task of

logging prediction. This work serves as a first step towards machine learning based catch-blocks and if-blocks logging prediction for Java projects.

Table 1: Experimental Dataset Details

Type	**Tomcat**	**CloudStack**	**Hadoop**
Version	8.0.9	4.3.0	2.7.1
No. of Java File	2036	5350	6331
Total Catch-Blocks	3279	8077	7947
Logged Catch-Blocks	887 (27%)	2792(34.56%)	2078(26.14%)
Distinct Exception Types	119	163	265
Total If-Blocks	16991	65392	32143
Logged If-Blocks	1423 (8.37%)	5653 (8.64%)	3407 (10.60%)

3.4 RO 4: Cross-project Logging Prediction Analysis

Cross-project logging prediction is a relatively unexplored area. Hence, as a first step toward cross-project logging prediction, we perform a large scale study of cross-project catch-block logging prediction on Java projects. We explore effectiveness of nine machine learning classifiers for cross-project catch-blocks logging prediction. We carefully select algorithms belonging to different domains such as probabilistic, decision tree etc. We answer several RQ's. We first compare performances of different machine learning algorithms for within-project and cross-project logging prediction. We then analyze performance of algorithm with respect to different parameters in order to identify whether they provide complementary information or not and finally compare the performances of single-project and multi-project training models. The output of this analysis is beneficial for building a robust cross-project logging prediction model.

3.5 RO 5: Ensemble Based Cross-project Logging Prediction

During cross-project model building, we face two main challenges. First, non-uniform distribution of numerical attributes, second, vocabulary mis-match problem. In order to address these challenges, we perform standardization of the attributes and also propose uses of ensemble based learning. Ensemble based learning is useful in improving the accuracy of base machine learning algorithms but their effectiveness with respect to cross-project logging prediction is unexplored yet. As a first step towards effective cross-project logging prediction, we propose ***ECLogger***, a novel ensemble based model for cross-project catch-blocks logging prediction. ECLogger uses 9 base machine learning algorithms and three ensemble techniques. We create 8 EClogger$_{bagging}$ models, by applying bagging on 8 base machine learning algorithms. We create 466 ECLogger$_{AverageVote}$ models by applying average vote ensemble technique on every possible combination

of base machine learning algorithms, using group of 3-9 algorithms at a time. Similarly, we create 466 ECLogger$_{MajorityVote}$ models using majority vote ensemble technique. In this work, we answer two RQ's. First, we compare the performances of baseline machine learning classifiers with the ECLogger classifier for cross-project catch-blocks logging prediction individually for each source and target project pair. Second, we compare the performance of baseline machine learning classifiers with ECLogger classifiers for cross-project catch-blocks logging prediction average on all source and target project pairs. We believe that this work serves as a first step towards improving cross-project logging prediction performance for Java projects.

4 Results

In this section, we give brief detail about the experimental dataset used for this study and the metrics used for prediction model evaluation. We then present results obtained for each RO.

4.1 Experimental Dataset

We select three large open-source projects from Apache Software Foundation (ASF) for evaluation: Tomcat, CloudStack, and Hadoop. Apache Tomcat is a web server that implements Java EE specifications like Java Servlet, Java Sever Pages and Java EL. CloudStack provides public, private, and hybrid cloud solutions. CloudStack provides a highly available and scalable Infrastructure as a Service (IaaS) cloud computing platform for deployment and management of networks of virtual machines. Hadoop is a framework that enables distributed processing of large datasetset. Table 1 shows details of the experimental dataset. All the three projects used in our study are long lived Java projects having several years ($\approx$ 7 to 17 year) of history. Table 1 shows that all three projects have several thousands of SLOC. Tomcat, CloudStack and Hadoop have been previously used by the research community for logging and other studies [3][8][17][23].

4.2 Metrics

In this subsection, we describe the performance metrics used to evaluate the effectiveness of the prediction models. At the time of prediction, we count the total number of logged code constructs predicted as logged ($C_{l \to l}$), logged code constructs falsely predicted as non-logged ($C_{l \to n}$), non-logged code constructs predicted as non-logged ($C_{n \to n}$), and non-logged code constructs predicted and logged ($C_{n \to l}$). Using these 4 values, we compute the following metrics:

$$Logged\ Precision\ (LP) = \frac{C_{l \to l}}{C_{l \to l} + C_{n \to l}} \times 100 \tag{1}$$

$$Logged\ Recall\ (LR) = \frac{C_{l \to l}}{C_{l \to l} + C_{l \to n}} \times 100 \tag{2}$$

$$Logged\ F-measure\ (LF) = \frac{2 \times LP \times LR}{LP + LR} \times 100 \tag{3}$$

$$Accuracy\ (ACC) = \frac{C_{l \to l} + C_{n \to n}}{C_{l \to l} + C_{l \to n} + C_{n \to n} + C_{n \to l}} \times 100 \tag{4}$$

Area under the ROC curve (RA): RA measures the likelihood that a positive class instance is given a high likelihood score compared to a negative class instance. RA can take any value in the range 0 to 1, higher the better.

Table 2: Results of survey performed on software developers from open-source projects

Q1	**How many years of experience do you have in software development?**
	⊙Less than 1 year (0%) ⊙Between 1 and 5 years (25.53%) ⊙More than 5 years (76.47%)
Q2	**Do you believe where and what to log is subjective?**
	⊙Strongly Disagree (11.76%) ⊙Disagree (11.76%) ⊙Neutral (17.65%) ⊙Agree (47.06%) ⊙Strongly Agree (11.76%)
Q3	**Fatal, Debug, Error and Info are various log levels. Do you believe assigning right verbosity level for a give case is challenging and developers make mistakes in verbosity level assignment?**
	⊙Strongly Disagree (0%) ⊙Disagree (17.65%) ⊙Neutral (25.53%) ⊙Agree (25.53%) ⊙Strongly Agree (35.29%)
Q4	**Do you believe a static code or program analysis tool or checker which can guide a developer in recommending where to log and which log level to use will be useful?**
	⊙Strongly Disagree (05.88%) ⊙Disagree (35.29%) ⊙Neutral (25.53%) ⊙Agree (29.41%) ⊙Strongly Agree (05.88%)
Q5	**In your experience, the modification and churn of logging code is:**
	⊙Rare (47.06%) ⊙Between Rare and Common (35.29%) ⊙Common (17.65%)

4.3 RO 1: Survey of Open-source Software Developers

We received a total of 17 responses of the survey. Among the 17 respondents more than 75% of the respondents have more than 5 years of experience in software development. Questions 2-4 are Likert questionnaire items in which the respondents specify their level of agreement or disagreement (ordinal variable) on a symmetric agree-disagree scale consisting of 5 choices. 60% of the respondents believe that where and what to log is subjective. 60% of the respondents believe that assigning right verbosity level for a given case is challenging and developers make mistakes in verbosity level assignment. 15% of the developers responded that in their experience, the modification and churn of logging code are common.

For Question 4, we conduct a Chi-Squared Test (non-parametric test) to test the goodness of fit between an expected frequency distribution (equal distribution between agree and disagree) and the observed frequency distribution. In the statistical significance testing, we get the p-value as 0.78. Since the p-value is greater than 0.1, we have no presumption against the null hypothesis (that a static code or program analysis tool or checker which can guide a developer in recommending where to log and which log level to use will be useful). The results of the survey reveal that more than 33% of the respondents believe that a checker to assist developers in automated logging can be useful.

> Software developers face challenges in source code logging and logging prediction tools can be beneficial.

4.4 RO 2: Empirical Analysis of Logged and Non-logged Code Constructs

We answer all the 9 research questions related to two level empirical analysis of logged and non-logged code constructs by conducting experiments on Tomcat, CloudStack, and Hadoop project (refer to Table 1). Our ***statistical-analysis*** of files shows that very less percentage of files are logged. For example, only 14.9% of files are logged in Tomcat project. Our analysis reveals that SLOC of logged files is considerably higher as compared to non-logged files and there exists a positive correlation between SLOC of a file and its log statement count. However, we notice presence of some very large non-logged files. The manual analysis reveals that these files are tool generated and hence do not consist of any log statements. ***Statistical-analysis*** of try-blocks show that for some projects try-blocks associated with logged catch-blocks have higher complexity as compared to that of non-logged catch-blocks. Our analysis reveals that few try-blocks i.e., $\leq$1.5%, consists of both logged and non-logged catch-blocks. We also observe that logging ratio of exception types is project specific. For example, exception type 'IOException', has logging ratio of 37.25%, 66.81%, and 27.72% for Tomcat, CloudStack, and Hadoop project, respectively. ***Content-analysis*** of try-blocks associated with logged and non-logged catch-blocks revels presence of different topics. For example, 'thread.sleep' topic is present in the non-logged catch-blocks of Tomcat project. Manual analysis reveals that in 84 occurrences of 'thread.sleep' it occurred 71 times in try-blocks associated with non-logged catch-blocks. More details on two level empirical analysis of logged and non-logged code constructs can be found in our published work [10].

> Empirical analysis reveals presence of different features in logged and non-logged code constructs at both the levels.

Table 3: LF for catch-blocks and if-blocks logging prediction using RF classifier

Model	Type	Tomcat	CloudStack
LogOpt	**Catch-blocks**	85.50%	93.4%
LogOptPlus	**If-blocks**	80.70%	92.25%

4.5 RO 3: With-in Project Logging Prediction

We empirically analyze all the proposed Boolean & numeric features and evaluate *LogOpt* and *LogOpt* models with five machine learning algorithms on two datasets (Tomcat and CloudStack). Results show that Random Forest (RF) classifier give the highest LF for both catch-blocks and if-blocks logging prediction for both the projects. RF classifier gives the highest LF of 93.4% and 92.25% for catch-blocks and if-blocks logging prediction (refer to Table 3). For both catch-blocks and if-blocks model gives the better results on CloudStack project. We analyze some of the exception types in both Tomcat and CloudStack project. Our analysis reveals the there are certain exception types in CloudStack project which are heavily non-logged. We believe that this is the reason for model performing better on the CloudStack project as compared to the Tomcat project for catch-blocks logging prediction. More details on logged code constructs prediction can be found in our published work [12] and [11].

> Machine learning based models are effective in catch-blocks and if-blocks logging prediction on Java projects.

4.6 RO 4: Cross-project Logging Prediction Analysis

We conduct our cross-project experiments on all the three projects (Tomcat, CloudStack, and Hadoop). We create six source & target project pairs by considering one project as source project and other two projects as target project (one at a time). Results show that performance of machine learning algorithms degraded considerably for cross-project catch-blocks logging prediction. We notice upto 6.37% to 18.05% drop in performance of machine learning algorithms for cross-project prediction as compared to within-project logging prediction. We also notice that performance of single-project and multi-project training models is similar. However, we notice that different classifiers are complementary to each other. For example, for CloudStack→Tomact project pair, ADTree algorithm gives the highest LP, ACC and RA values, whereas BN gives the highest LF. Hence, combination of different machine learning algorithms can be useful in improving cross-project catch-blocks logging prediction performance. More details on cross-project logging prediction analysis can be found in our published work [9].

Different classifiers provide complementary information and hence ensemble based approach can be effective in improving the cross-project catch-blocks logging prediction performance.

4.7 RO 5: Ensemble Based Cross-project Logging Prediction

We evaluate performances of *ECLogger* models on all the three projects (Tomcat, CloudStack, and Hadoop). For each source and target project pair, we report the ECLogger model giving the best results. We observe that ensemble based models are effective in improving the cross-project catch-blocks logging prediction performances. Results show that average vote based ECLogger model performs the best and give highest results (both individual as well as average performance) as compared to the baseline classifier. Average vote model gives the highest improvement of 3.12% (average LF) and 6.08% (average ACC). Overall, we observe that CloudStack project is more generalizable as compared to the Tomcat and the Hadoop project for cross-project catch-block logging prediction. Manual analysis reveals that CloudStack project provide support for both Tomcat and Hadoop projects and hence gives better results for cross-project prediction. More details on ensemble based cross-project logging prediction can be found in our published work [9].

Ensemble based model is effective in improving the cross-project catch-blocks logging prediction performance.

5 Threats to Validity

In this section, we discuss various threats to validity to this work.

5.1 External Validity

External validity refers to the generalization of the results. In our work, we investigate three open source Java projects which have different domains and sizes. However, the results may not be generalizable to other types of project such as closed source or projects written in other programming languages as different projects may have different logging practices.

5.2 Construct Validity

The construct threat refers to how we identify log statements in the source code. We create 26 regular expressions to find all the log statements. The manual inspection reveals that we identify all the types of log statements. However, there is still a possibility that we missed some types of log statements.

5.3 Internal Validity

Threat to internal validity refers to the error in our code and bias in sampling. We have double checked our code to remove errors from our code. To mitigate the sampling bias in training and testing dataset split creation, we create 10 training and testing dataset and report the average results.

6 Conclusions and Future Work

This work aims towards analysis and prediction of log statements on Java projects. We start by performing a survey of software developers from open-source projects. Results of the survey revel that developers face challenges in logging. Now in order to provide software developers with better logging mechanism, we perform an in-depth, focused, and two level empirical study of logged and non-logged code constructs. Preliminary results show distinguishing features between logged and non-logged code constructs which can be helpful in predicting logged code constructs. Using inputs from our empirical study we propose machine learning based models for catch-blocks and if-blocks logging prediction on Java projects. Initial results show that both the models are effective for within-project logging prediction. We analyze performance of this catch-block logging prediction mode for cross-project logging prediction. Our results reveal that performance of model degrades considerable for cross-project logging prediction as compared to within-project logging prediction. We then propose an ensemble based approach to improve cross-project catch-block logging prediction performance. Results show that ensemble based approach is effective in improving cross-project logging prediction performance. Our initial results are encouraging but much work yet remains to improve within-project and cross-project logging prediction performance. In future, we plan to extend the current work for other types of code constructs such as while loops, switch statements. ECLogger model proposed in this thesis is used for cross-project logging prediction only for catch-blocks. Hence, can be extended to other types of code constructs such as if-block. We plan to extend this work for other language such as C#, Python. We also plan to study the evolution of log statements over the lifetime of projects.

Bibliography

[1] Blackberry enterprise server logs submission. `BlackBerryEnterpriseServerLogsSubmission`. [Online; accessed 4-June-2016].

[2] Java regains spot as most popular language in developer index. `http://www.infoworld.com/article/2909894/application-development/java-back-at-1-in-language-popularity-assessment.html`. [Online; accessed 19-March-2016].

[3] Boyuan Chen and Zhen Ming (Jack) Jiang. Characterizing logging practices in java-based open source software projects – a replication study in apache software foundation. *Empirical Software Engineering*, pages 1–45, 2016.

[4] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 24–33, 2014.

[5] Steven Haines. Top 5 performance problems in .net applications. `https://blog.appdynamics.com/net/top-5-performance-problems-in-net-applications/`. [Online; accessed 27-July-2016].

[6] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automatic identification of load testing problems. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 307–316, Sept 2008.

[7] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automated performance analysis of load tests. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 125–134, Sept 2009.

[8] Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, and Ahmed E. Hassan. Examining the stability of logging statements. In *The 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016.

[9] S. Lal, N. Sardana, and A. Sureka. Eclogger: Cross-project catch-block logging prediction using ensemble of classifiers. *eInformatica Software Engineering Journal*, 2017.

[10] Sangeeta Lal, Neetu Sardana, and Ashish Sureka. Two level empirical study of logging statements in open source java projects. *International Journal of Open Source Software and Processes (IJOSSP)*, 6(1):49–73, 2015.

[11] Sangeeta Lal, Neetu Sardana, and Ashish Sureka. Logoptplus: Learning to optimize logging in catch and if programming constructs. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 215–220, June 2016.

[12] Sangeeta Lal and Ashish Sureka. Logopt: Static feature extraction from source code for automated catch block logging prediction. In *9th India Software Engineering Conference (ISEC)*, pages 151–155, 2016.

[13] Michelle Levesque. Fundamental issues with open source software development. *First Monday*, 9(4), 2004.

[14] Heng Li, Weiyi Shang, and Ahmed E Hassan. Which log level should developers choose for a new logging statement? *Empirical Software Engineering*, pages 1–33, 2016.

[15] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 382–391, May 2013.

[16] Weiyi Shang, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Michael W Godfrey, Mohamed Nasser, and Parminder Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, 26(1):3–26, 2014.

[17] Weiyi Shang, Meiyappan Nagappan, and Ahmed E. Hassan. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering*, 20(1):1–27, 2015.

[18] X. Xia, D. Lo, S. McIntosh, E. Shihab, and A. E. Hassan. Cross-project build co-change prediction. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 311–320, March 2015.

[19] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 293–306, 2012.

[20] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 102–112, 2012.

[21] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. Improving software diagnosability via log enhancement. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 3–14, New York, NY, USA, 2011. ACM.

[22] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, M.R. Lyu, and Dongmei Zhang. Learning to log: Helping developers make informed logging decisions. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 415–425, May 2015.

[23] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM.

Hammouda, I., Lundell, B., Madey, G. and Squire, M. (Eds.) Proceedings of the Doctoral Consortium at the 13th International Conference on Open Source Systems, Skövde University Studies in Informatics 2017:1, ISSN 1653-2325, ISBN 978-91-983667-1-6, University of Skövde, Skövde, Sweden.

Skövde University Studies in Informatics 2017:1
ISSN 1653-2325
ISBN 978-91-983667-1-6

www.his.se

UNIVERSITY
OF SKÖVDE