EMPIRICAL ANALYSES OF SOFTWARE CONTRIBUTOR PRODUCTIVITY

by
Ayushi Rastogi

Under the supervision of
Dr. Nachiappan Nagappan and Prof. Pankaj Jalote

Indraprastha Institute of Information Technology, Delhi
August 2017

EMPIRICAL ANALYSES OF SOFTWARE CONTRIBUTOR PRODUCTIVITY

by
Ayushi Rastogi

Submitted
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
to the

Indraprastha Institute of Information Technology, Delhi
August 2017

Dedicated to my Mom, Dad and Di.

CERTIFICATE

This is to certify that the thesis titled "Empirical Analyses of Software Contributor Productivity" being submitted by *Ayushi Rastogi* to the *Indraprastha Institute of Information Technology, Delhi* for the award of the degree of Doctor of Philosophy, is an original research work carried out by her under our supervision. It is to be noted that from July 2011 to August 2014 research was carried out under the supervision of Dr. Ashish Sureka. In our opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

August 2017

Dr. Nachiappan Nagappan

Prof. Pankaj Jalote

Indraprastha Institute of Information Technology, Delhi
New Delhi, 110020

ACKNOWLEDGEMENTS

In the end, it's the journey that matters...

My PhD journey is beautiful because of the people in the forefront and in the background who stood by me. My advisers, collaborators and committee members stood at the forefront guiding me throughout and my friends/family acted as my invisible strength. As I finish my PhD, I would like to thank all those whose thoughtfulness, care and support helped me reach here.

Many thanks to my advisers Dr. Nachiappan Nagappan and Prof. Pankaj Jalote whose constant support and guidance made this thesis possible. I thank Dr. Ashish Sureka who introduced me to research during the initial years of my PhD and laid the foundation for my future works. Thanks to Dr. Thomas Zimmerman, Dr. Suresh Thummalapenta, Jacek Czerwonka, and Dr. Georgios Gousios for their collaboration and fruitful discussions. Also thanks to my PhD monitoring committee: Dr. Rahul Purandare and Dr. Pushpendra Singh, for their constant feedbacks on the thesis. I thank my thesis review committee: Prof. Premkumar Devanbu, Prof. Michele Lanza and Dr. Anjaneyulu Pasala for sharing their suggestions to improve the work. I also thank my industry mentor - Ravindra Naik and comprehensive examination review committee - Dr. Pamela Bhattacharya and Dr. Gautam Shroff for their inputs.

I am indebted to my Mom, Dad and Di who listened to all my crazy talks and helped me get back to track whenever I digressed from normal life. And my wonderful friends: Sangeeth, Monika, Ankita, Sonia, Megha, Surabhi, Dipto, Sonal, Hemanta, Annapurna, and Venkatesh within the institute and many more outside the institute, who made all these years memorable. Thank you all for being there.

I thank the TCS PhD research fellowship programme which supported me financially throughout my PhD (January 2012 to December 2016). Also thanks to Microsoft, CICS, SIGSOFT, TCS, and IIIT-Delhi for providing travel support.

Last but not the least, thanks to all the survey respondents from industry and open source projects for sharing their perspectives and experiences.

PUBLICATIONS

[1] Ayushi Rastogi, Nachiappan Nagappan, and Georgios Gousios. "Geographical Bias in GitHub: Perceptions and Reality." In: *Submitting to ICSE 2018*.

[2] Ayushi Rastogi and Nachiappan Nagappan. "On the Personality Traits of GitHub Contributors." In: *27th International Symposium on Software Reliability Engineering, Ottawa, Canada*. 2016.

[3] Ayushi Rastogi and Nachiappan Nagappan. "Forking and the Sustainability of the Developer Community Participation–An Empirical Investigation on Outcomes and Reasons." In: *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE. 2016, pp. 102–111. DOI: 10.1109/SANER.2016.27.

[4] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. "Ramp-Up Journey of New Hires: Tug of War of Aids and Impediments." In: *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2015, pp. 1–10. DOI: 10.1109/ESEM.2015.7321212.

[5] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. "Ramp-up Journey of New Hires: Do strategic practices of software companies influence productivity?" In: *10th Innovations in Software Engineering Conference (ISEC), formerly India Software Engineering Conference*. 2017, pp. 1–4.

[6] Ayushi Rastogi. "Contributor's Performance, Participation Intentions, Its Influencers and Project Performance." In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. 2015, pp. 919–922. DOI: 10.1109/ICSE.2015.292.

[7] Ayushi Rastogi and Ashish Sureka. "What Community Contribution Pattern Says about Stability of Software Project?" In: *21st Asia-Pacific Software Engineering Conference*. Vol. 2. 2014, pp. 31–34. DOI: 10.1109/APSEC.2014.88.

[8] Ayushi Rastogi and Ashish Sureka. "Open Source Software: Mobile Open Source Technologies: 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Proceedings." In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. Chap. Does Contributor Characteristics Influence Future Participation? A Case Study on Google Chromium Issue Tracking System, pp. 164–167. ISBN: 978-3-642-55128-4. DOI: 10.1007/978-3-642-55128-4_22. URL: http://dx.doi.org/10.1007/978-3-642-55128-4_22.

[9] Ayushi Rastogi and Ashish Sureka. "SamikshaViz: A Panoramic View to Measure Contribution and Performance of Software Maintenance Professionals by Mining Bug Archives." In: *Proceedings of the 7th India Software Engineering Conference*. ISEC '14. Chennai, India: ACM, 2014, 2:1–2:10. ISBN: 978-1-4503-2776-3. DOI: 10.1145/2590748.2590750. URL: http://doi.acm.org/10.1145/2590748.2590750.

[10] Ayushi Rastogi, Arpit Gupta, and Ashish Sureka. "Samiksha: Mining Issue Tracking System for Contribution and Performance Assessment." In: *Proceedings of the 6th India Software Engineering Conference*. ISEC '13. New Delhi, India: ACM, 2013, pp. 13–22. ISBN: 978-1-4503-1987-4. DOI: 10.1145/2442754.2442757. URL: http://doi.acm.org/10.1145/2442754.2442757.

[11] Ayushi Rastogi and Ashish Sureka. "SamikshaUmbra: Contribution and Performance Assessment of Software Maintenance Professionals by Mining Software Repositories." In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. Vol. 2. 2013, pp. 170–175.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Software is everywhere and touches all aspects of human lives in one form or the other. The relevance of software in today's world can be simply gauged by its current and ever increasing market size. According to the industry analyst Gartner, worldwide software market size in 2016 is expected to be US$326 billion[1] with a 5.3% growth from the previous year 2015. The relevance of the industry and its tremendous impact makes it important to optimize software productivity.

Software productivity is impacted by three main dimensions - process, people and tools/technology [1]. Much of the earlier works on productivity improvement focused on process and tools/technology. Studies on process focused on individual processes [2], overall process of a project or an organization for improvements [3]. Studies on tools/technology evaluated the usefulness of tools/technology for improving software productivity [4]. Relatively less focus was given to people aspects.

People are important assets in software projects as software productivity depends on the productivity of the contributors. This realization dates back to 1970s when Fred Brooks in 'The Mythical Man-Month' argued that technical issues are subordinate to human issues while explaining the quality of software development [5]. Further, Gerald Weinberg in 'The Psychology of Computer Programming' addressed "programming as human performance and social activity" [6]. However, there is not much information to explain how software contributors perform their tasks in workplace [7]. Recently, the studies in software engineering – an intensely people-oriented activity, observed a rise in explaining software productivity in terms of contributor productivity.

The objectives of the studies on contributor productivity are to completely utilize the capabilities of contributors and improve contributor productivity. To utilize the capabilities of contributors, studies measure contributor productivity [8][9][10]. These measures of contributor productivity are useful for performance appraisal, identify expertize, generate team awareness, identify strengths and areas of improvements for training, etc. On the other hand, to improve contributor productivity, studies try to understand the influence of factors like initial environment [11], diversity [12], etc. on contributor productivity. This information is useful to promote best practices and mitigate or eliminate the negative influences of factors.

One approach for contributor productivity studies is based on the experiences of seasoned software professionals. The experience reports, processes, models, etc. generated by these professionals are valuable and provide a deeper understanding of the contributor productivity. In modern software organizations, practitioners often rely on prior experiences, intuitions and gut feelings gathered from their stint in software development for decision making [13]. However, this approach fails to provide evidence, deep models, and analytical tools to inform decisions. Also, as software continues to grow (in size, complexity, etc.) and evolve (in terms of tools, technologies, etc. being used) rapidly, it becomes imperative to explain contributor productivity based on well-formed claims. There is a need of a transparent, data-driven and objective way to substantiate software contributor productivity. These insights can be realized through empirical studies.

Empirical studies provide a way to move towards well-founded decisions. It compares what we believe to what we see and is capable of providing a deeper level of understanding of the software development. This thesis uses Empirical Software Engineering approaches based on Mining Software Repositories to help improve contributor productivity through the quantitative and qualitative analyses of the data collected and generated during software development.

This thesis empirically analyses software contributor productivity to identify best practices and areas for improvements. The empirical evidence gathered in these studies enable decision makers to promote best practices and take corrective or preventive measures to mitigate or eliminate the negative effects of areas for improvements. The informed decisions supported by empirical evidence have applications in improving software contributor productivity and software productivity, in general.

This thesis looks at software contributor productivity from several angles. Primarily, this thesis investigates the influence of some factors on software contributors' productivity, but in a related way, also proposes measures of software contributor productivity. These measures of software contributor productivity are used to provide a holistic understanding of developer contributions as an individ-

1 http://www.gartner.com/newsroom/id/3186517

ual and in a team to identify outlier contribution patterns arising from factors such as malpractices. This holistic understanding facilitates productivity improvements by generating awareness about the prevailing good or bad developer contribution patterns.

In this chapter, we first give a brief background of Empirical Software Engineering, Mining Software Repositories and Software Contributor Productivity. The background information of Empirical Software Engineering and Mining Software Repositories is gathered from various sources, primarily the books: 'Guide to Advanced Empirical Software Engineering' [7], and 'Basics of Software Engineering Experimentations' [14], and research papers: 'Future of Empirical Methods in Software Engineering Research' [15], 'Selecting Empirical Methods for Software Engineering Research' [16], 'Empirical Studies of Software Engineering: A Roadmap' [17] and 'The Road Ahead to Mining Software Repositories' [13]. This is followed by an overview of the contributions of this thesis, research method used, and details of each contribution.

1.1 EMPIRICAL SOFTWARE ENGINEERING

Empirical studies validate the existing beliefs in software engineering by providing evidence [14]. These studies explore, describe, explain, predict phenomenon using evidence from observation or experience [15]. Empirical studies start with specifying a research question. Based on the type of research question asked, a study design is proposed. To execute the proposed design, required data or evidence are gathered which are then analyzed for interpretation [15]. Broadly there are two approaches to conduct empirical studies: quantitative analysis and qualitative analysis.

Quantitative research applies mathematical models on numerical data to get formal results [7]. It collects numerical data and looks for relationships between the variables under examination [7]. For instance, quantitative study to analyze the impact of two tools on programmer productivity measures changes in programmer productivity.

Qualitative research, on the other hand, is done with words by observing objects in their natural settings [7]. The observations are then combined to compare, contrast, analyze and identify patterns [7]. The intention of conducting qualitative research is to get a holistic overview of the context under study [7]. For instance, qualitative study to analyze the influence of two tools on programmer productivity examines the closeness of the logics of the tools to human reasoning.

Quantitative investigations get more justifiable and formal results than qualitative inquiries [7]. However, qualitative studies are necessary for comprehensively defining the full body of knowledge of any discipline [7]. Quantitative and qualitative analysis are complementary in nature [7]. Importantly, the concept of subjectivity and objectivity are not necessarily correlated with either of these types of investigations [7].

1.1.1 *Research Methods*

The choice of method for a study depends on various factoring like availability of resources, access to subjects, opportunity to control variables of interest, etc [16]. Further, a variety of methods can be applied to any research problem, where a combination of methods are necessary to fully understand the problem. Below we describe the five classes of research methods that are most relevant to software engineering [16]:

Controlled Experiments (including Quasi-Experiments)

A controlled experiment investigates a testable hypothesis to look for a cause-effect relationship [16]. In this experiment, independent variables are manipulated to measure the effect of dependent variable by keeping variables other that the chosen independent variables as constant - control variables [16]. These experiments use an initial random assignment and are conducted in strictly controlled conditions [16].

A close variant of controlled experiments is Quasi-controlled experiments, where "subjects are not assigned randomly to treatments" [16]. Quasi-experiments are used where true experiments are not possible [16]. For instance, in fields study. In Quasi-controlled experiments, the comparison happens between non-equivalent groups [16]. Thus, the task of interpreting quasi-experiments is separating the effects of a treatment from those due to the initial incomparability between the average units in each treatment group [18]. Here, the strength of the evidence is related to the degree of control we have in the studies we perform [16]. Quasi-experiment is less powerful that true experiment and requires more careful interpretations [16].

The strength and limitation of this technique is that it is theory driven [16]. The good thing about this approach is that we do not search for results [16]. Conversely, since we build on theory, we might ignore variables which might hold relevance outside laboratory setting [16].

Case Studies (both exploratory and confirmatory)

Case study is an "empirical enquiry that investigates a contemporary phenomenon within its real-life context especially when the boundary between the phenomenon and context are not clear" [16]. They offer an in-depth understanding of the mechanism by which cause-effect relationship occurs [16].

There are two types of case studies: Exploratory case study and Confirmatory case study [16]. Exploratory case studies are used as "initial investigations of some phenomenon to derive new hypotheses and build theories" [16], while Confirmatory case studies "test existing theories" [16]. Confirmatory case studies are important for refuting theories and choosing between rival theories [16].

For conducting a case study it is important to have "a clear research question of how and why a certain phenomenon occurs" [16]. Case study is most appropriate where the context is expected to play a role in the phenomenon and hence controlled experiments are inappropriate [16].

A major weakness of case study is that the collection and analysis of data is eligible for various interpretation and researcher bias [16]. Hence, an explicit framework is required for the selection of cases and data collection [16]. Also, while individual case study often reveals deep insights, the validity of the results depends on the subsequent case studies that support or refute the theory [16].

Survey Research

Survey research "identifies the characteristics of a broad population of individuals" using questionaire, structured interviews or data logging [16]. Two key things related to survey research are 1) selecting representative sample from a defined population and 2) data analysis techniques used for generalization [16]. A precondition for conducting survey research is a clear research question that explores the nature of target population [16]. To select a representative subset of samples, stratified sampling techniques are used to ensure that no one community is being over-represented [16].

The challenge of conducting survey research is to control for sampling bias which decides the generalizability of the results [16]. Also, low response rates increase the risk of bias [16]. Other challenging thing is to design questions in a way that yields "useful and valid data" [16]. Phrasing questions that is understandable to a diverse population is difficult [16]. Moreover, there is a possibility that what people do and what they say are different due to their inability to introspect reliably on their work practices [16]. So, it is generally encouraged to compare survey results with other empirical methods [16].

Ethnographies

Ethnography focuses on field observations to "study a community of people to understand how the members of the community make sense of their social interactions" [16]. This helps to understand "how technical communities build a culture of practices and communication strategies that enables them to perform technical work collaboratively" [16].

Ethnography does not build on any theory, rather it creates local theories using the description of the community to understand the culture of the community [16]. For such studies, researchers have to explicitly consider their pre-conceptions [16].

A precondition for such studies is a research question with focus to understand the "cultural practices of a community and access to the members of that community" [16]. To identify target members, usually representative members are identified and chain sampling is used to select further members [16].

A special form of ethnography is participant observation where researchers become member of the community for the duration of analysis [16]. For this the researcher should have the required technical competency to become a member of the project and requires a much longer duration for the study [16].

The biggest challenge for ethnographic research is to avoid preconceptions while performing observation, data collection and analysis [16]. Researchers are expected to obtain a high degree of training to conduct such studies [16].

Action Research

In action research, researchers attempt to solve a real problem by intervening in the studied situation for improvements [16]. A precondition for action research is to have a problem owner – most of the time the researcher themselves, who identifies the problem and tries to solve it [16]. The researcher is expected to critically reflect "upon their past, current and planned actions to identify what helped (or not) to solve the problem" [16].

Two key criteria for conducting this study is judging the authenticity of the problem and knowledge outcomes for the participants [16]. The key characteristic of this research method is to bring real changed and solve the problem iteratively [16]. Importantly, no attempt is being made to establish a control group, rather the focus is to identify helpful lessons [16].

The biggest challenge for action research is its immaturity as an empirical method because of the lack of frameworks for their implementation [16]. Finally, this method may be expensive due to the required commitment from the organization [16].

Mixed-Methods Approach

Mixed-methods research originates from the stance that "all methods have limitations and that the weakness of one method can be compensated for by the strengths of other methods" [19]. This approach uses both – quantitative and qualitative data collection and analysis techniques [16].

Mixed methods approach is powerful, however, the researcher is required to 1) collect extensive data, 2) analyze multiple sources of data and 3) requires familiarity with both quantitative and qualitative forms of research [16]. The three most commonly used strategies for conducting mixed-methods research are:

1. *Sequential explanatory strategy* This strategy is characterized by "collection and analysis of quantitative data followed by the collection and analysis of qualitative data" [16]. The purpose of this strategy is "to use qualitative results to assist in explaining and interpreting the findings of a quantitative study" [16]. It is useful when unexpected results arise [16].

2. *Sequential exploratory strategy* This strategy is characterized by "collection and analysis of qualitative data followed by the collection and analysis of quantitative data" [16].The purpose of this strategy is "to use quantitative data and results to assist in the interpretations of qualitative findings" [16]. It is useful to examine a proposed theory formulated as the outcome of a qualitative study [16].

3. *Concurrent triangulation strategy* This strategy is characterized by using "different methods concurrently to confirm, cross-validate or corroborate findings" [16]. This strategy is motivated by the fact that 'what people say' and 'what people do' could be different [16]. So, in this strategy researchers collect data from multiple sources helps improve validity [16].

1.1.2 *Data Collection Techniques*

Based on the levels of interaction with software engineers, the data collection techniques can be classified as direct, indirect and independent [7]. Selecting an appropriate technique is influenced by the question asked and the amount of resources available to conduct the study.

- *Direct*: Researchers directly interact with the participant population [7]. Direct data collection techniques can be further classified as inquisitive and observational based on the type of information gathered [7]. Inquisitive techniques help obtain a "general understanding of the software engineering process" [7]. Examples of inquisitive techniques are brainstorming, focus groups, interview, questionaries, etc.

  Observational techniques provide a "real-time portrayal of the studied phenomenon" [7]. This technique is subjective and requires considerable knowledge to interpret correctly [7]. Further, this technique always run the risk of Hawthorne effect [7]. Examples are: participant observation, shadowing and observation, think-aloud sessions, work diaries, etc.

- *Indirect*: Researchers have access to the participants via direct access to their work environment [7]. There is no need of direct contact between the participant and the researcher [7]. The data is automatically gathered once the data collection is initiated [7]. This technique involves little or no intervention of participants and is ideal for longitudinal studies [7]. For example, instrumenting systems, fly on the wall, etc.

- *Independent*: Researchers have access to the work artifacts of the software developers which is used to uncover information about how software engineers work [7]. These artifacts are generated as a by-product of software development and can act as a primary source of information [7]. For example, source code, documentation, etc.

Independent data collection techniques

Independent data collection techniques helps "uncover information about how software engineers work by looking at their output and by-products" [7]. In this thesis, we mostly use independent data collection techniques. We briefly discuss the various types of independent data collection techniques and the advantages/disadvantages of analyzing them.

Analysis of electronic databases of work performed

Most large organizations manage the work performed by developers using issue tracker, problem reporting, change request, and configuration management system [7]. The data generated by such systems are a rich source of information for software engineering researchers [7]. Below are the advantages and disadvantages of analyzing these databases [7]:

- Advantage
  - A large amount of data is readily available for analysis.
  - The data is stable and is not influenced by the presence of researchers.
- Disadvantage
  - There is very little control over the quantity and quality of information manually entered about the work performed.
  - It is difficult to gather additional information.

Analysis of tool logs

Software systems generate logs in various forms [7]. For instance, automatic building tools leave records as source code control system.

- Advantage
  - The data is already in electronic form for coding and analysis.
  - The logged behavior is part of software engineering normal work routine.
- Disadvantage
  - Companies use different tools in different way, so it is difficult to gather data consistently across organizations.

Documentation Analysis

This technique focuses on "the documentation generated in the source code as well as documents describing the software system" [7]. Other sources of documentation are local newsgroups, group e-mail lists, memos, documents defining the development process, etc.

- Advantage
  - Documents capture conceptual information about the system. It also acts as an introduction to the software and team. It provides low-level information of algorithms and data.
- Disadvantage
  - Reading documentation is time consuming and requires a level of understanding.
  - Documentation may be inaccurate.

Static and Dynamic Analysis of the System

This technique analyzes “the code (static analysis) or traces generated by running the code (dynamic analysis) to learn design and understand how software engineers work” [7].

- Advantage
  - Readily available and large amount of information is ready to be mined.
- Disadvantage
  - Need parser and other tools to extract useful information from source code, which in its current form are not of high quality.

One field of research that analyzes historical software development activities to conduct empirical studies is Mining Software Repositories. The field of Mining Software Repositories automates various activities associated with the empirical studies in software engineering. This automation facilitates replication on a large number of projects which helps generalizing the findings and validating the theories.

1.1.3 *Mining Software Repositories*

Software development is facilitated by various tools. These tools generate artifacts that are archived for future references. These artifacts are termed as software repositories. Software repositories are static record-keeping repositories that are maintained centrally to manage progress of software projects [13]. Software repositories contain wealth of valuable information about projects. The information stored in software repositories provide data-driven and objective measures based on historical and field data and thus relies less on intuition and experience [13].

Mining Software Repositories (MSR) describe the broad class of investigation into the examination of software repositories. MSR field “analyzes and cross-links the rich data available in these repositories to uncover interesting and actionable information about software systems” [13]. MSR researchers aim to transform these repositories from static record-keeping ones into active repositories which can guide decision processes in modern software projects [13]. So going by the management adage *"What you cannot measure; You cannot control"*, Mining Software Repositories provide data-driven evidence in objective and transparent manner.

The focus of mining software repositories is in [13]:

- Techniques to extract information from repositories.
- Discovery and validation of novel techniques and approaches to mine information.

Types of Repositories

There are three types of software repositories: historical, run-time and code repository [13]. Below we discuss the three types of repositories in details:

HISTORICAL REPOSITORIES Historical Repositories contains “several information about the evolution, progress and current state of the project” [13]. For instance, bug repositories (also known as Issue Tracking System), source control repositories, peer code review system, version control system and archived communications (such as mailing lists, threaded discussion and IRC).

1. *Bug Repositories* also termed as defect tracking system keeps a track of reported software bugs in software development projects. It is a type of issue tracking system. For instance, Google Chromium Issue Tracking System, Bugzilla, Jira, etc.
2. *Version Control Repositories* This repository records the development history of a project. It tracks all the changes to the source code along with the meta-data about each change. For instance, Git, Mercurial, Subversion, Perforce, etc.
3. *Peer Code-review System* A system which is used to examine the documentation, code, etc. by authors and one or more colleagues to evaluate its technical content. For example, Rietveld, Gerrit, GitLab, etc.
4. *Archived Communications* This repository tracks the communications in a team involved in software development. For example, mailing lists, IRC chats, instant messages, etc.

RUN-TIME REPOSITORIES Run-time repositories contain "information of the execution and usage of an application at one or multiple deployment sites" [13]. The availability of these logs is on rise due to their use of remote issue resolution and legal acts. For instance, deployment logs, build warnings, test results, etc.

CODE REPOSITORIES Code Repositories contain "source code of various applications developed by several developers" privately or publicly [13]. They are often used by open source software and multi-developer projects to handle versions. These websites often support version control, bug tracking, release management, mailing lists, and wiki-based documentation. For instance, Google Code, SourceForge, PicoForge, GitHub etc.

Strengths and Limitations of Mining Software Repositories

One good thing about the MSR field is that the cost of experimenting with MSR techniques is usually low as the data is readily available. However, there are some limitations of mining the repository data [13]:

- Repository data cannot be used to conclude causation instead it can only show correlation.
- Findings must be investigated closely within the context of the studied project or system. Project and system context are very important to reveal the true case for particular findings. For example, some developers are more likely to perform buggy changes for the reason that they are usually assigned more complex changes and not due to the skill level of these developers.

Thus, the limitations of the repository data should be closely examined and communicated when presenting results to avoid misinterpretations.

1.1.4 *Impact of Empirical Software Engineering*

The applications of Empirical Software Engineering encompass the phases of software development to address a wide range of problems. Starting from requirement analysis to maintenance of software development, empirical software engineering provide solutions to correctly specify requirements [20], enable correct design decisions [21], efficient and quality software development [22], cost-effective maintenance of software [23], etc.

Studies using Empirical Software Engineering provide solutions from various perspectives. From process perspective, it helps understand differences in actual and observed processes, bottlenecks in development [24], etc. It helps curate the usability and effectiveness of tools and techniques in various contexts of software development [25]. It helps understand how people with distinctive characteristics interact with the team and environment to perform [11]. Empirical studies also provide tools that help people in their work [26]. A combination of these diverse perspectives helps towards a better understanding of software development for quality product, satisfied workforce, efficient management, etc.

1.2 SOFTWARE CONTRIBUTOR PRODUCTIVITY

Besides technical improvements, attempts to improve software productivity are directed towards *utilizing* and *improving* contributor productivity. These two objectives form the basis of studies on software contributor productivity. Studies on utilizing software contributor productivity estimate the potential of contributors in terms of their development activities and use this information for assigning work, providing necessary training, compensation, etc. Another class of study looks into factors that facilitate or inhibit participation and use this information to plan actions to improve software contributor productivity. Thus approaches to improve software contributor productivity can be broadly classified into two classes: measure contributor productivity and understand contributor productivity. Below we discuss the various perspectives with which studies measured and explained contributor productivity. The list of perspectives presented in this thesis is not exhaustive and is closely related to the research contributions of this thesis.

1.2.1 *Measuring Contributor Productivity*

Assessment of contributions is a standard practice in organization that serves versatile needs of various roles. Managers or decision-making roles inspect contributions to identify expertize, areas of improvements, resource management, individual assessment, team assessment, recruitment, promotions; study adversarial behavior, turnover, retention plans, training programs, etc. On a contrary, contributors look to this information for self-reflection, team awareness, etc. Below we describe some of the applications of measurement of contributor productivity.

Evaluation

Evaluating contributor productivity is important for self-reflection, performance appraisal, identify strengths, areas of improvements, etc. Traditionally estimating software contributor productivity for evaluation was based on the experiences of managers and self assessment of individuals. To bring objectivity to these measures, researchers realized the potential of mining software repositories. Existing studies mined a variety of software repositories [27][28][29] and proposed simple measures indicating contributor participation [8][30]. The proposed measures also accounted for the distributed development environment [31] and used these measures in conjugation with the traditional software development activities [32] to evaluate contributors.

Fernandes et al. proposed an analytical model and conducted a practical case-study in context of distributed software development setting [33]. Kanij et al. pointed out the lack of well established metrics for software testers and conducted a survey of industry professionals to identify important factors for measuring performance of software testers [34]. Kidane et al. proposed two productivity indices: creativity index and performance index for online communities of developers and users of open source projects such as Eclipse [35]. Gousios et al. presented a list of pre-defined developer's actions for measuring developer's involvement and activity by mining software repositories such as source code repository, document archives, mailing lists, discussion forums, defect tracking system, Wiki and IRC [36]. Nagwani et al. proposed a team-member ranking technique for software defect archives and generate a ranked list of developers based on individual contribution [37]. Gilbert et al. explored group dynamics to compare developers and their contribution in a distributed software community by analyzing social visualization code [9].

Authorship Analysis

In large collaborative software projects, it is important to identify who did what to identify expert for a task, generate team awareness, plagiarism detection, etc. Authorship analysis includes author discrimination, author characterization and similarity detection. Author discrimination identifies which of the known set of authors have written a new source code. Author characterization associates contributor characteristics with the source code. Finally, similarity detection analyzes the degree of overlap in code.

Existing studies proposed several numeric metrics [38] and visualizations to analyze authorship [9]. Taylor et al. introduced the concept of author entropy to characterize authorship. Author entropy in conjunction with other software metrics has the potential to identify areas of concerns within source code [38]. Krein et. al further introduced the concept of language entropy to characterize the distribution of an individual's development efforts across multiple programming languages [39]. Further, more subjective elements of authorship based on fuzzy logic linguistic variables in combination with objective counts improves authorship analysis [40].

Various visualization tools are proposed for authorship analysis. CVSscan tool observes the source code structure and evolution during software maintenance. This tool colors source code according to author, age or code construct [25]. StatSVN generates charts, tables and statistics for software development. It shows contributions of authors at directory level [41]. CodeSaw [9] visualizes author activity in distributed software development by combining code activity and developer communication. This tool reveals group dynamics by presenting individual and team contributions.

Team Activity Awareness

To generate awareness of the activities of fellow contributors and to get a holistic view of team activities various visual presentations are proposed. Existing studies presented various types of visualizations namely spatial [10], temporal visualizations [42], etc. They analyzed tasks and social graphs [42] and devised ways to analyze collaborative development [43]. Other studies semantically aggregated tool feeds and presented the results [44].

Storey et al. proposed a framework to describe, compare and understand human centric awareness visualization tool in software development [26]. SeeSoft tool provides a line-based view of the code where rows are colored to represent a particular aspect of the code file [45]. CodeThumbnails provide a miniature view of the spatially arranged code. The active sections of the files are highlighted [46]. TeamTracks keeps a record of code navigations in the past to enable programmers to build on the actions of others [47]. FastDash provides a real-time awareness of the activities of fellow contributors [10].

Several tools convey the status of shared artifacts. Community Bar tool allows collaborators to see shared resources [48]. SEAPort extends this capabilities to multiple shared and personal devices [49]. CoWord [50] and Network Text Editor [51] faciliate collabortive work within a document.

Project Stability

A large number of developers leave the project in subsequent releases. Packages left orphan (resulting from developer leaving the project) are either adopted (by current working developers) or lost. Packages thus lost are not included in stable release leaving users unsupported for their unique functional needs [52]. Robles et al. added time dimension to onion model (organizational structure of libre software projects) to study joining and leaving process in organization. A quantitative study of 21 large, well-known projects shows that majority of libre software projects have several generations of developers (over time) and very few projects are led by founding core groups [53]. Robles et al. proposed a novel methodology to visually analyze evolution of core team to ensure smooth transitions by identifying breakpoints and unevenness [54].

A majority of open source software projects fail due to their inability to garner significant developer participation. Various researches analyzed the causes of turnover and its impact. Hall et al. conducted an empirical study of 89 software practitioners to identify causes of turnover in software projects that hampers project success. Results suggest that increase in motivation of developers reduces staff turnover [55]. Sharma et al. proposed a Logistic Hierarchical Linear Model approach to analyze turnover in open source software based on individual developer level factors and project specific factors. Analysis showed significant variations in turnover rate among projects with past activity of developer, developer role, project size and project age as prime indicators [56]. Schilling et al. suggested that team integration (measured as conversations in mailing list) facilitates contribution retention in open source software projects. They expressed team integration as a function of spatial, temporal and cultural distances of open source software developers and validated the results using subjective and objective means [57].

1.2.2 *Understanding Contributor Productivity*

The productivity of software contributors can be explained in terms of their willingness, capacity and opportunity [58]. Willingness to perform is influenced by the motivation, job involvement, personality, feelings of equity of contributors, etc [58]. Age, ability, knowledge, level of education, etc. – variables of capacity are seen to influence contribution [58]. Similarly, variables of opportunity: working conditions, actions of coworkers, leader behavior, mentorism, organizational policies, time, etc. are seen to influence contribution [58]. In this section, we discuss the factors that are seen to influence software contributor productivity, where contributor productivity is measured as contributions and participation intentions.

The work by Zhou et. al showed the influence of willingness, capacity and opportunity on the long term contribution intentions of new comers [59]. They found that during first month, long term contributors show more community-oriented attitude. They found that new comers who start by commenting or reporting an issue, that is fixed, are more likely to be long term contributors. They found that a very popular project and less attention from peers reduces the chances of participation while working with clustered and productive peers increases the chances of future participation.

Open source development draws from a diverse set of motivation, many of which are based on external rewards. Internal factors like intrinsic motivation, altruism, and identification with a community influences the motivation for participation in open source projects [60]. External factors like direct compensation and anticipated returns influence the motivation for participation in open source

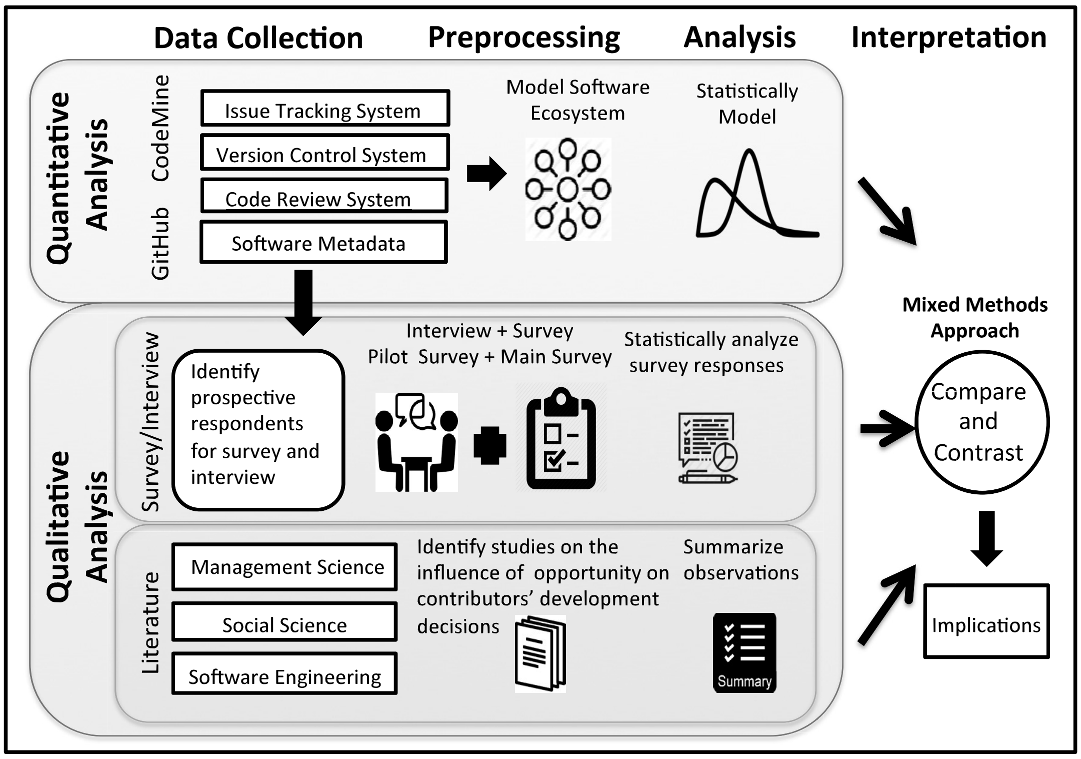


Figure 1: Research Method - Mixed methods approach comparing and contrasting the observations of quantitative analysis and qualitative analysis

projects [60]. Factors that promises future monetary rewards and personal need for a software solution were other key factors. Another study showed that a software developer's ability and individual need for achievement are the two strongest factors determining individual performance [61]. A longitudinal study on Apache projects showed that a sense of obligation to the community and use value also influences participation [62]. This study proposed a theoretical model and evaluated it on Apache projects. Person-job and person-team fit also explained the participation behavior [63].

Other studies examined the influence of demographic attributes: gender and tenure on productivity [64], explored gender bias in evaluating contributors in GitHub [65]. The influence of communications [66], personality [67], spatial, temporal, cultural distance [68][57], diversity [12] and various other project and contributor characteristics [56][59] on contributor productivity are studied.

1.3 THESIS OVERVIEW

This thesis derives its results from detailed studies on employee productivity at Microsoft, participation of contributors in GitHub, and measures of contributions of Google Chromium software maintenance team. The study of large popular product groups at Microsoft reveal factors that influence the ramp-up journey of new hires. The studies on GitHub analyze the dynamics of competing projects, personality traits, and perception on bias to explain the participation of contributors. Also, the studies on Google Chromium project propose measures of contributions for software maintenance team.

Each chapter in this thesis captures a different perspective with which we can help improve the productivity of software contributors. The chapter-wise contributions of this thesis are:

1. Chapter 2 presents a comprehensive list of aids and impediments in the ramp-up journey of new hires at Microsoft. The results of this study acts as a guide for managers to ensure that their new employees reach the productivity levels of existing employees faster. Further, it helps the faculty in academia to understand the skills needed in a software industry [69] [70].

2. Chapter 3 looks for the potential presence of bias which influences the sense of equity and hence developer participation. This chapter presents an analysis of the perceptions and reality around the presence of bias in peer code review in GitHub. The results of this study generate awareness among software contributors that there exists a bias in the peer code review process which is perceived to be experienced by the contributors and not the reviewers [71].

3. Chapter 4 explains the influence of various contributor and project characteristics on contributor participation in open source projects. This chapter facilitates managers in evaluating the sustainability of their project based on the dynamics of its competing projects. It also helps managers understand the intricacies of the team by explaining the personality types that contributes more and are best suited for a role. It also provides information on the the role of contributor characteristics on their participation [72] [73] [74].

4. Chapter 5 provides robust, data-driven and role-based measures of contributor productivity in the maintenance team of Google Chromium project. This study provides measures of individual and team contributions and facilitates the analysis of contributions in the context of software development. These measures and visualizations help contributors in self-reflection and managers in evaluating contributors, identifying strengths, areas of improvements, outlier behavior, etc. [75] [76] [77] [78].

This thesis uses a mixed-methods approach [19][16] to take advantage of the unique strengths and mitigate the limitations of quantitative and qualitative analysis. Figure 1 summarizes the approach used in this study.

To understand software development activities we gather the experiences of software professionals and model the software development activities by mining software archives and its associated metadata. We analyze data in various software repositories like Issue Tracking System, Version Control System, etc. to model the software ecosystem and test the hypotheses statistically. To get a deeper understanding of the phenomenon, we conduct survey or interview and build on literature.

For survey or interview, we select respondents with desired characteristics by analyzing software repositories or personal relations. We conduct a small scale interview or a pilot survey and incorporate suggestions and feedbacks received to modify the main survey. The main survey is sent to all prospective survey respondents. The results of the survey are statistically analyzed for interpretation.

The other approach is to gather insights from literature. Studies in management science, social science and software engineering explains contributor productivity. We use observations from these domains to formulate our initial hypotheses or cross-validate the observations. We compare and contrast the observations from quantitative analysis and qualitative analysis to draw inferences.

The contributions of this thesis are summarized in Figure 2 and are briefly discussed below:

1. Factors Influencing the Ramp-up Journey of New Hires - Chapter 2

Hiring top talent is essential for any software company's success. After joining the company, new hires often spend weeks or months before making any major contribution and attaining the same productivity level as existing employees. Ramp-up journey is the transition of new hires from novice to experts. There can be several factors, such as lack of experience or lack of familiarity with processes unique to the new company, which influence the ramp-up journey. To understand such aids and impediments in the ramp-up journey, we analyzed data extracted from version control systems of eight large and popular product groups in Microsoft with several thousand software developers. In particular, we study two aspects of the ramp-up journey. First, the time taken to make first check-in into the version control system, an important milestone in the ramp-up journey indicating the first contribution. Second, the time taken to reach the same productivity level as existing employees in terms of check-ins. Further, the results of quantitative study is augmented with qualitative results derived by surveying 411 professional developers. Our study produces promising results, including factors such as unavailability of resources, having a mentor, prior knowledge of required skill sets, proactively asking questions, career stage, etc. that could help improve the ramp-up journey of new hires.

Factors ranging from 'unavailability of resources' to 'career stage' influence the ramp-up journey of new hires.

2. Presence of Geographical Bias in Peer Code Review Process - Chapter 3

Open source development has often been considered to be a level playing field for all developers. But there has been little work to investigate if bias plays any role in getting contributions accepted

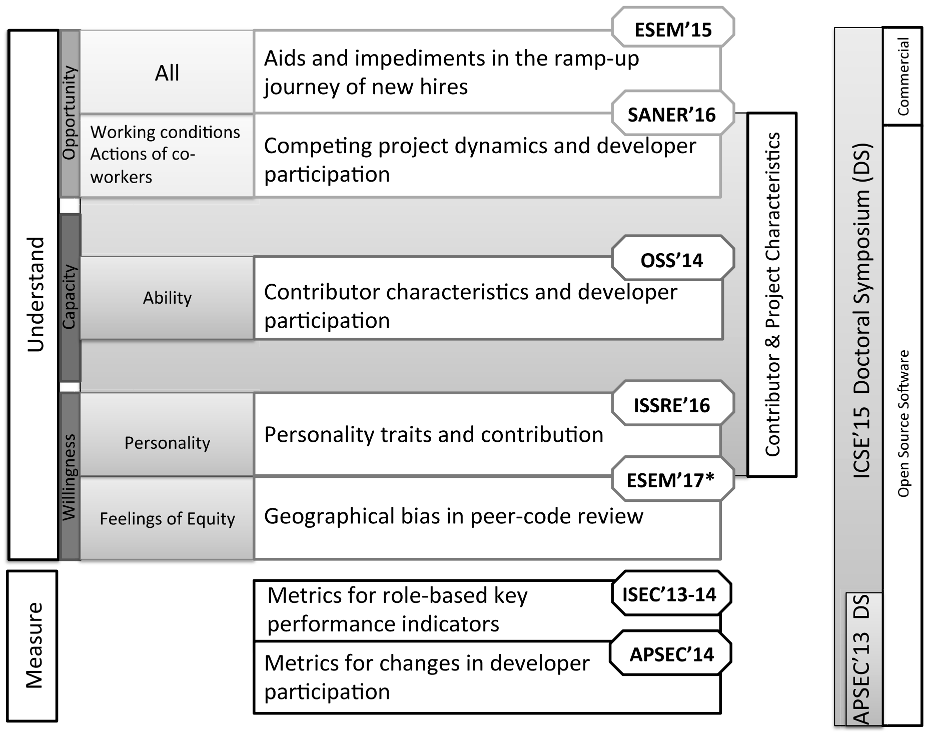

Understand
Opportunity
Capacity
Willingness
All
Aids and impediments in the ramp-up journey of new hires
ESEM'15
Working conditions Actions of co-workers
Competing project dynamics and developer participation
SANER'16
Ability
Contributor characteristics and developer participation
OSS'14
Personality
Personality traits and contribution
ISSRE'16
Feelings of Equity
Geographical bias in peer-code review
ESEM'17*
Contributor & Project Characteristics
Measure
Metrics for role-based key performance indicators
ISEC'13-14
Metrics for changes in developer participation
APSEC'14
ICSE'15 Doctoral Symposium (DS)
APSEC'13 DS
Commercial
Open Source Software

Figure 2: Overview of the research contributions of this thesis

and developers' perceptions of the bias. We present a study - one of the largest of its type, to understand the perceptions and reality of the influence of geographical location on the evaluation of pull requests in GitHub - one of the primary open source development platforms. Using a mixed-methods approach that analyzes 70,000+ pull requests and 2,500+ survey responses, we find a bias blind spot. Data analyses showed that geographical location explains differences in pull request acceptance decisions. Compared to the United States, submitters from the United Kingdom(22%), Canada(25%), Japan(40%), Netherlands(43%), and Switzerland(58%) have higher chances of getting their pull requests accepted. However, submitters from Germany(15%), Brazil(17%), China(24%), and Italy(19%) have lower chances of getting their pull requests accepted. The probability of pull request acceptance increases by 19% when the submitter and integrator are from the same geographical location. Survey responses from submitters indicate that the perceptions of bias are strong in Brazil and Italy, matching the data analysis. However, integrators do not perceive as being biased.

A bias blind spot exists in the peer code review process in GitHub.

Project and Contributor Characteristics on Contributor Participation - Chapter 4

Contributor participation is a function of the willingness, capacity, and opportunity of contributors and their interactions. Existing studies analyzed various factors along the three dimensions and studied their influence on contributor participation. For instance, studies explained the influence of willingness on contributor participation in terms of intrinsic motivation, extrinsic motivation, feeling of fairness, etc. The influence of capacity on contributor participation was explained in terms of knowledge, age, personality, etc. Similarly, initial environment, demographics, etc., - variables of opportunity of contributors are seen to influence contributor participation. Our studies bridge few gaps in literature to analyze the influence of some project and contributor characteristics in explaining contributor participation. Particularly, we explain the rise and fall of developer participation due to competing projects, the effect of personality traits on levels of contributions, and the impact of role reputation and contribution on developer participation.

3. Rise and Fall of Developer Participation due to Competing Projects

A majority of Open Source Software projects fails due to their inability to garner significant and sustained developer community participation. The problem proliferates when competing projects emerge from the source code of an existing project, a phenomenon called forking of the original project, claiming existing and potential developer community participation. Our study empirically analyze the influence of forking on the sustainability of the developer community participation in the original project. Further, it explains the observed behavior in terms of the characteristics of the project observed at the time of forking. A large-scale study of 2,217 projects hosted on GitHub shows that 1 in every 5 original projects observes a decline in the sustainability of the developer community participation after forking. The negative effect is more pronounced in projects ported to GitHub from other platforms (~20%), compared to GitHub developed projects (~9%). Also the observed behavior can be explained in terms of the characteristics of the competing projects at the time of forking. For instance, in medium sized projects an increase in the maturity of the original project by a year decreases the odds of decline in the sustainability of the developer participation by 23%.

One in five projects observe decline in developer participation when competing projects grow. This rise and fall of developer participation can be explained in terms of the characteristics of competing projects.

4. Effect of Personality Traits on Levels of Contribution

People's personality has the potential to explain the behavior in different contexts. Our study explores the inferential power of personality traits in explaining the behavior of contributors in various contexts of software development in GitHub. Analyses of 243 actively discussed projects showed that the contributors with high or low levels of contributions are more neurotic compared to the contributors with medium level of contributions. Analyses of 423 active contributors showed that contributors evolve as more conscientious, more extrovert and less agreeable over the years of participation. The findings of this study match our intuitions and are promising for further explorations.

Certain personality traits are associated with higher levels of contributions. Personality traits of active contributors evolve with time. Personality traits of contributors are context-specific.

5. Impact of Role Reputation and Contribution on Developer Participation

Understanding and measuring factors influencing future participation is relevant to organizations. This information is useful for planning and strategic decision-making. Our study measures contributor characteristics and compute attrition to investigate their relationship by mining Issue Tracking System. The experiments are conducted on the four years data extracted from Google Chromium Issue Tracking System. Experimental results show that the likelihood of future participation increases with increase in relevance of the role in project and level of participation in previous time-interval.

Role reputation and contribution is directly proportional to the likelihood of developer participation.

Measures of Contributor Productivity - Chapter 5

Measuring contributor productivity is important to identify expertize, generate awareness, identify strengths and areas of improvements, etc. Our studies propose metrics for individuals and team to help analyze contributions and team stability respectively.

6. Measure Individual Contribution

Individual contribution and performance assessment is a standard practice conducted in organizations to measure the value addition by various contributors. Accurate measurement of individual contributions based on pre-defined objectives, roles and key performance indicators is a challenging task. Further, appraisers discern the need to scrutinize underpinning factors involved in performance appraisal. Our study presents 'Samiksha' - a framework for contribution and performance assessment of software maintenance professionals by mining software repositories. The framework provides 11 role-based contribution and performance assessment metrics and demonstrate the results on real-world data of Google Chromium Issue Tracking System. The requirement gathering of factors influencing performance is based on survey conducted on experienced software maintenance professionals. Further, we propose a framework for visualization - 'SamikshaViz' to study panoramic view of individual's contribution and performance in a team. These visualizations reinforce and extend the rationale of the metrics proposed in 'Samiksha' and gather detailed insights for appraisers to justify performance. The results are validated by practitioners in industry.

Performance indicators perceived important by managers are not measured in practice. Role based metrics based on key performance indicators are useful.

7. Measure Team Contribution Patterns

Community management is challenging in Open Source Software projects as contributor participation is fluid. Contributor churn (joining or leaving a project) causes failure of the majority of software projects. We present a framework to characterize stability of the community in software maintenance projects by mining Issue Tracking System. Our study identifies key stability indicators, proposes metrics to measure them, conducts time series analysis on metrics data to examine the stability of the community and models community participation patterns to forecast future participation. A case study of Google Chromium project investigates the inferential ability of the framework.

Metrics to measure temporal developer participation help analyze stability of project.

2 AIDS AND IMPEDIMENTS IN THE RAMP-UP JOURNEY OF NEW HIRES

Hiring top talent is essential for any software company's success. After joining the company, new hires often spend weeks or months before making any major contribution and attaining the same productivity level as existing employees. *Ramp-Up journey* refers to this transition of new hires from novice to experts. There can be several factors, such as lack of experience or lack of familiarity with processes unique to the new company, which influence the ramp-up journey. To understand such aids and impediments in the ramp-up journey, this chapter provides a study by analyzing data extracted from version control systems of eight large and popular product groups in Microsoft with several thousand software developers. In particular, we studied two aspects of the ramp-up journey.

First, we studied time taken to make the first check-in into the version control system, an important milestone in the ramp-up journey indicating the first contribution. Second, we analyzed the time taken to reach the same productivity level as existing employees in terms of check-ins. We further augmented our quantitative study with qualitative results derived by surveying 411 professional developers. In this Chapter, we describe the work and results obtained. Our study produced promising results, including factors such as having a mentor, prior knowledge of required skill sets and proactively asking questions, that could help improve the ramp-up journey of new hires.

2.1 INTRODUCTION

Hiring new talent is one of the core competencies of any software company, to meet evolving business requirements and to stay ahead of its competitors. For instance, over the past decade Microsoft hired several thousand software developers each year. These new hires range from fresh college graduates without any prior industry experience to professional developers with several years of experience. Soon after on-boarding the company, new hires undergo rigorous training in getting familiar with not only their assigned project, but also with processes and the overall culture of the company. Therefore, new hires take some time before making any contribution to their project and becoming as productive as existing employees. We use the term *ramp-up journey* to refer to this time period spent by new hires in transitioning from novice to experts and becoming as productive as existing employees.

There exists several reasons why ramp-up journey of new hires can span up to several weeks or months. In the case of college graduates, despite of best curricula there are still gaps between what graduates learn in the college and what they need to know to be productive in a typical work environment [79][80]. For instance, graduates are trained in various skills such as programming and development methodologies (Agile or Extreme Programming). However, college graduates often lack communication and teamwork skills, and are also not prepared to deal with various aspects such as complex development processes, legacy code, and tight deadlines [81]. On the other hand, experienced professionals, although familiar with some aspects such as working under tight deadlines, also face challenges in a new company. The primary reason is that different companies use different processes, technologies, tools, etc. For example, IBM primarily uses Rational Team Concert [82] as a version control system, whereas Microsoft primarily uses Source Depot, Team Foundation Server [83], or Git [84]. Therefore, experienced professionals need to master new processes and technologies before becoming as productive as existing employees [85].

To understand the aids and impediments in the ramp-up journey of newly hired software developers, we conducted a large-scale study that uses mixed data analysis. Our analysis combines software engineering data extracted from version control systems and qualitative data from surveys and interviews. We analyzed recent releases of eight large and popular product groups in Microsoft with several thousand software developers. These eight product groups account for the majority of engineering workforce at Microsoft. We observed that 14-49% of all software developers in product teams considered in our study are new hires (refer Figure 3). In Figure 3, the horizontal axis shows different product groups and the vertical axis shows percentage of new hires for the duration of product release under analysis. We use P1 to P8 to identify different product groups and to anonymize the results for confidentiality. Percentage of new hires in product groups as high as 49% makes it interesting to understand factors that influence the time it takes for new hires to become productive. To

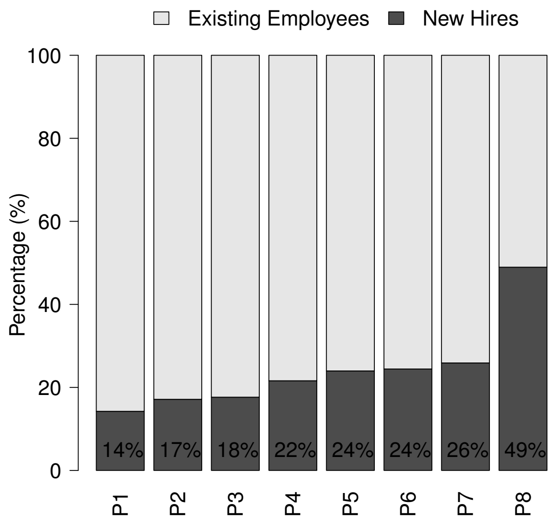


Figure 3: Percentage of new hires in different product groups

gain deeper understanding of the factors that influence the ramp-up journey of new hires with different experience levels, we used three levels to represent new hires: entry, middle, and senior. Entry level represents developers such as college graduates without any prior industry experience. Middle level represents developers with 1-3 years of experience or have higher qualification such as doctorate degree. Finally, senior level represents developers with more than 3 years of experience.

We analyze the ramp-up journey of new hires on two aspects. The first milestone in the ramp-up journey of new hires is achieved when new hires make first check-in into the version control system. Specifically, we examine check-ins in the master branch or shippable branch. The choice of master or shippable branch ensures that all check-ins analyzed in the study mark significant contribution and are not test check-ins. Thus, first check-in ensures that new hires have attained a basic understanding of the engineering system used by the project and also some basic knowledge of the project to achieve the task. Finally, new hires ramp-up when they attain the productivity level of existing employees. In this study, we measure the time to first check-in and the ramp-up time to analyze the productivity of new hires and understand factors that influence the ramp-up journey.

Our motivation is that the results of our study help fix the problems from a two pronged approach for both industry and academia. From the industry perspective, this information help managers and business analysts fix some of the bottlenecks in the existing processes and improve useful practices to help new employees ramp-up faster, increase morale, improve productivity, etc. From an academic perspective, the study can help faculty understand the skills needed for the students to succeed in the industrial environment. To summarize, in this study, we try to answer two broad sets of research questions:

1. The factors that influence the ramp-up journey of new hires; and

2. The amount of time it takes for new hires to become productive.

To the best of our knowledge, our study is the first that attempts to combine both quantitative and qualitative data to address these questions. There have been prior studies [86][87][88] discussed in related work that approach this from different perspectives like interviews, surveys, etc. However, we are the first to quantitatively combine both software engineering data and data from developers' opinions. Noteworthy to add that some of the findings from this study may already be known in the industry, however, there exists no empirical evidence drawn from a systematic analysis of software engineering data. We believe that product teams can use the results from this study to adopt best practices.

Our results indicate that the ramp-up journey is influenced by various factors in the company. We discovered that while having a mentor, prior knowledge of required skill sets, etc. help in increasing the productivity; lack of proper documentation, trying to get access and permissions, etc. reduce the productivity of new hires. We complete the story by presenting a comprehensive list of activities that new hires engage in and present their suggestions to improve the productivity.

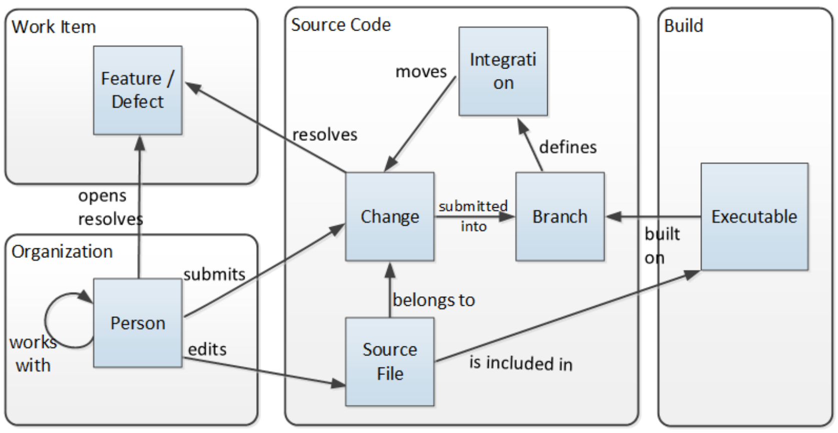


Figure 4: Data collected by CodeMine

2.2 RELATED WORK

There exists a large body of research on integrating freshly recruited university graduates into the organization. Begel and Simon conducted a qualitative analysis of fresh university graduates. They observed that a large fraction of problems encountered by university graduates is due to their inexperience with the corporate environment [86]. Dagenais et al. provided an initial theory on project landscapes to help new hires familiarize faster with the team [87]. Their study emphasized the relevance of mentor and good documentation to help new hires adjust and perform in team. Sim and Holt interviewed new hires to identify patterns in which they familiarize themselves with the project and the environment, and discussed its implication [89]. Besides these, Ostroff investigated the role of mentoring in the learning process of newcomers [88]. The author observed that newcomers with mentor have a better understanding of organizational issue and practices compared to others. Another class of study provide recommendations to bridge gaps between the understanding of college graduates and the requirements of the industry. Raderwacher and Walia conducted a systematic literature review to identify the most common areas of deficiency in university graduates from academia or industry job perspectives [90]. Similarly, Begel and Simon observed difficulties in the transition from college graduates to experienced software engineers and suggested changes in curricula and software engineering courses[80]. Brechener goes on to recommend courses that might help bridge the expectation gaps between academia and industry [79]. The existing studies have largely focused on understanding problems encountered by fresh university graduates and attended this question from the qualitative analysis perspective. In relation to existing studies, our study makes the following two novel contributions:

1. We study the ramp-up journey of new hires ranging from fresh university graduates to professionals with prior job experience.

2. We quantitatively analyze the software engineering data and augment the results with qualitative analysis.

2.3 BACKGROUND

CodeMine [91] provides a data collection framework for all major Microsoft development teams. It collects information from source code repositories, irrespective of their format, and stores data on changes, sources, branches, and code integrations in a normalized schema. It does the same for builds, work item repositories, and organizational data. After normalizing into a common schema (see Figure 4), it creates relationship between artifacts. For example, it creates relationship between work items and source changes to capture the changes made in response to a work item. Similarly, it creates relationship between builds and source code to capture the changes which appeared for the first time in a particular build. When the relationships are built, CodeMine exposes all collected and interpreted data as a service. As a result, the tool is ideal for learning about the practices used across teams and developing a set of metrics that can be used for generally characterizing branch structures.

Table 1: Products and start-end date of the releases

Products	Start Date	End Date
Azure	2011-01-01	2013-12-31
Bing	2009-12-30	2013-11-12
Exchange	2010-10-01	2013-08-31
Office	2011-09-01	2013-08-31
SQL Server	2009-07-01	2012-03-06
Windows	2009-10-22	2013-07-01
Windows Phone	2010-07-03	2014-02-03
Windows Server	2009-10-22	2013-07-01

2.4 METHODOLOGY

2.4.1 *Data Collection*

In this section, we describe the data collection methodology for our study. We use a mixed data analysis technique combining software engineering data extracted from version control systems and qualitative data from surveys and interviews, to answer our research questions. Table 1 summarizes the products from which we analyzed new employees and the start-end date of the releases under analysis.

To determine the 'new hires' in our analysis, used in the rest of the study, the employee in the software development role had to satisfy one or more of the following four criteria:

1. The employee is a university recruit or this is the first company of the employee.
2. The employee joined the company as an intern or vendor and converted to a full time position.
3. The employee left the company and joined again after at least a year.
4. The employee worked for other companies in the past.

In point 3, we consider returning employees as new hires because they have to adapt to the rapidly changing technology and practices in the company, in addition to the other factors. The rest of the software developers, including internal transfers, are termed as existing employees for this study. Next, we explain below the quantitative and qualitative analysis methodology.

2.4.2 *Quantitative Analysis*

The core part of our quantitative analysis is formed by the CodeMine data [91]. We use the version control system database and the employee information database as the starting point in our analysis. In the ramp-up journey, for new hires to transition from novice to experts, they have to attain the productivity level of existing employees. To do so, we first define a baseline for productivity of software developers in the context of our study.

Software developers perform various activities like writing code, reviewing code, debugging code, etc. The objective of these activities is to generate useful features, provide fixes to existing bugs, etc. In Microsoft, all software developers are expected to make code check-ins into the version control system as part of developing new features or making bug fixes. So, one denomination to measure the contribution of software developers is code check-ins. In this study, we examine various factors related to code check-in to suggest the productivity of software developers.

The first milestone in the ramp-up journey of new hires is when they make their first check-in into the version control system. First check-in marks the first useful contribution, and time to first check-in indicates the time it takes to make a useful contribution. Since, the first check-in alone does not suggest that the developer is productive, we also measure the time to ramp-up. We quantify the ramp-up time as the time it takes for a new hire to reach the average productivity level of existing employees. To measure the time to ramp-up, we examine the frequency of check-ins as an indicator of the familiarity with the process that comes with initial experience. Thus, frequency of check-ins measure the familiarity with the process. However, it falls short to comment on the efforts and the span of knowledge required to implement the change. To capture these features, we examine artifacts related to source code like lines changed and files changed. Lines changed indicate efforts of software developers, where the contribution can be in the form of code, comments, or documentation. Similarly, files changed indicates that the developer has acquired an understanding of a set of files and their relationships. These measures, that is, check-ins count, lines of code changed, and files changed have also been used in literature to measure developer's contribution [92][93].

In this section, we measure the time to first check-in and examine the ramp-up time using frequency of check-ins, lines changed, and files changed. Here, it is important to note that the three metrics used to compute ramp-up time are not exhaustive. However, the three metrics capture the key contributions of software developers and give a fair picture of the ramp-up journey. In addition to this, we examine the influence of 1) experience, 2) product team of the new hire, 3) proximity to the core team and 4) internship within the same company on the time to first check-in and the ramp-up time. In this study, we try to answer the following research questions:

RQ1 Does the product group of new hires influence the time to first check-in?

RQ2 Does prior job experience, within or outside the company, influence the time to first check-in?

RQ3 Does proximity to the core team influence the time to first check-in?

RQ4 Does internship within the company influence the time to first check-in?

RQ5 Does early check-in correlates with early ramp-up?

RQ6 Is ramp-up journey a function of experience and product?

RQ7 Is ramp-up journey a function of proximity to the core-team and internship within the company?

2.4.3 *Qualitative Analysis*

To complete the understanding of factors that influence the ramp-up journey of new hires, we augment quantitative analysis with qualitative results. Here, qualitative results include a small set of interviews and a broad deployment of a survey directed towards new hires at Microsoft. To get a flavor of factors that influence the ramp-up journey of new hires and to help with designing the survey, we conduct a small scale interview of four software developers. Three out of the four interviewees were 'Entry Level' software developers and one of them was 'Senior Level' software developer. We interviewed software developers for a half hour each and presented them with two open-ended questions.

1. What factors supported or undermined their attempts to make early first check-in and reduced the time to ramp-up?
2. What could have been done to reduce the time to first check-in and the ramp-up time?

We use the responses from the four interviews to formulate two sets of multiple-choice questions, in addition to the demographics and open ended questions. The first set of questions try to understand the influence of specified factors on the time to first check-in. Similarly, the second set of questions try to understand the effect of specific factors on the time to ramp-up. The two sets of questions are evaluated on a 5-point Likert scale, along with an additional field titled 'I don't know'. The field 'I don't know' is intended to address cases where the survey respondents have no understanding of the situation described as factors influencing the ramp-up journey. Both sets of questions are followed by an open ended question to capture factors that are not mentioned in the list. Next, we asked software developers the list of activities, other than code check-in, that demands their time and efforts. Finally, we requested their suggestions on practices that may help reduce the ramp-up time.

To conduct the survey, we identified software developers from the eight product groups based on the following criteria:

- Developers have some minimum experience with the aids and impediments that can be faced during ramp-up.
- The aids and impediments encountered during the ramp-up journey are still fresh in their minds.
- Have a reasonable sample of new hires, as we anticipated a response rate of $\approx 20\%$ based on previously conducted studies.

Table 2: Statistical significance of survey results

Parameter	Increase	No effect	Decrease	p-value
Lack of proper documentation for the project	267	63	75	$<0.001^{***}$

Based on the above criteria, we identified 1,189 software developers with 6 to 13 months of experience at Microsoft on the date of analysis. The survey was sent to all identified software developers representing different roles, career stage paths, and nationalities. We received 411 completed responses (34.57% response rate), 1 partially filled response, and no disqualified response. 99.8% of the responses we received were from individual contributors. The other roles asked in the survey were lead and manager.

As we present the qualitative results, we first analyze the usefulness of the summaries of survey by computing statistical significance using chi-square test at 0.05 significance level. To compute statistical significance, we convert the ordinal scale to its nominal equivalent. We merge 'Strong Increase' with 'Moderate Increase' and collectively present the result as 'Increase'. Similarly, we merge 'Strong Decrease' with 'Moderate Decrease' as 'Decrease'. The rest two categories, 'I Don't Know' and 'No Effect' are considered as one and are titled 'No effect/I don't know'. For each statistically significant result, we compute the central tendency as the most frequent response or 'mode' and present the results. Table 2 presents a sample question asked in the survey that tries to understand the impact of 'Lack of proper documentation for the project' on the time to first check-in. A detailed discussion of the complete list of factors is given in the next two sections. In Table 2, 267 survey respondents said that lack of proper documentation for the project increases the time to first check-in. 63 survey respondents either did not encounter this problem or considered that it had no effect, and 75 responses suggested that it decreases the time to first check-in. We conducted chi-square test and observed p-value<0.001. P-value<0.001 gives a very strong presumption against the null hypothesis, thereby suggesting that the result is statistically significant. The claim made by central tendency summaries, as identified by mode, is that *'Lack of proper documentation increases the time to first check-in'*.

In the next two sections, we present an analysis of the time to first check-in and the ramp-up time respectively. We quantitatively analyze the time to first check-in and the ramp-up time, followed by the summaries of the multiple choice questions asked in the survey. Further, to complete the analysis, we present the opinions of survey respondents presented in the open ended questions. We present the list of activities that claim the time and efforts of new hires and also suggestions from the survey respondents to improve the ramp-up journey. We card sort the opinions of the new hires in the open-ended questions and present it in the non-increasing order of the frequency of occurrence.

2.5 TIME TO FIRST CHECK-IN

The time to first check-in marks the first step in the ramp-up journey of new hires. We measure the time to first check-in as the duration from the starting date at Microsoft until the time the new hire makes first check-in into the master branch of the version control system. For confidentiality reasons, we obscure the unit of time used in the study and report the results.

2.5.1 *Quantitative Analysis*

RQ1: Does the product group of new hires influence the time to first check-in?
Microsoft is a large software company with multiple product divisions. Each product division at Microsoft is slightly different from other divisions in terms of tools, technologies, or processes being used. Here, we are interested to know whether working with some specific product group helps new hires make early first check-in into the system. By answering this question, we can identify some of the best practices in product groups which can then be transferred to other product groups. To do so, we measure the time to first check-in for all new hires in the eight product teams and compute quartiles. The choice of quartiles for the study ensures that our results are not affected by outliers. Figure 5 shows the boxplot of the time to first check-in for the new hires in the eight product teams. Here, the horizontal axis shows the product groups and the vertical axis shows the time to first check-in measured in weeks. In Figure 5, we observe that the median of the population of new hires across all product divisions take ≈4-10% of the maximum time to first check-in. The analysis of eight product groups at Microsoft suggests that working in some specific product group has no significant

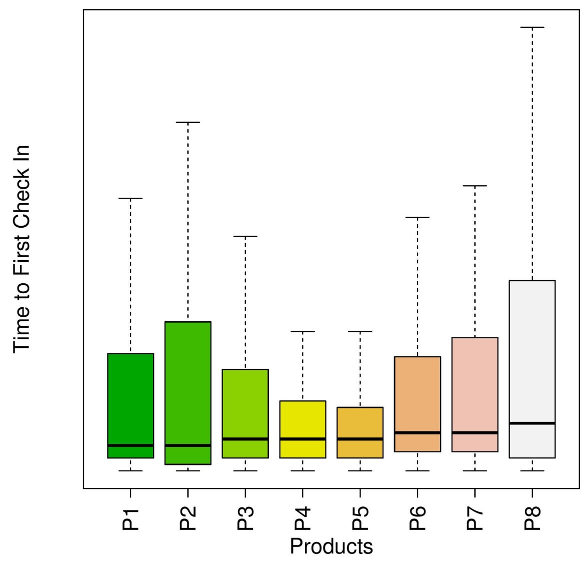


Figure 5: Product groups and time to first check-in

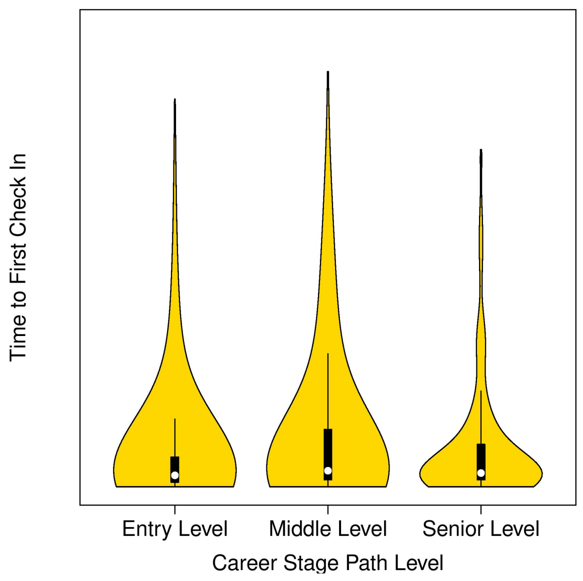


Figure 6: Experience and time to first check-in

impact on the time to first check-in. However, we see that the third quartiles of the product teams show marked differences. Figure 5 shows that new hires in products P2 and P8 take longer (≈34% and ≈43% of the maximum time to first check-in across products respectively) relative to other product groups (minimum ≈14% in product P5). Further investigation is required to understand the cause.

RQ2: Does prior job experience, within or outside the company, influence the time to first check-in? Microsoft recruits several thousand new hires every year, ranging from fresh university graduates to professionals with prior job experience. Here, we are interested to know whether prior job experience help new hires make early first check-in into the system. To investigate the impact of prior job experience on the time to first check-in, we analyze the influence of the career stage path or job title, an indicator of experience, on the time to first check-in. At Microsoft, professionals are hired with job-titles that conform to their prior job experience. For instance, professionals with 1 to 3 years of experience are hired as 'Middle Level' software developers. Thus, in this study, we analyze the role of job title on the time to first check-in. To start with, we classify job titles in Microsoft into 'Entry Level', 'Middle Level' and 'Senior Level' software developers. 'Entry Level' include software developers with job titles 'Software Development Engineer (SDE)' and 'IT SDE'. 'Middle Level' include job titles 'SDE 2' and 'IT SDE 2', and 'Senior Level' include job titles 'Senior SDE', 'Principal SDE', 'Partner SDE' , and 'Distinguished Engineer'. For each career stage path, we compute the time to first check-in for all new hires and plot the results.

Figure 6 shows the viola plot where the horizontal axis shows the three career stage paths of software developers and the vertical axis shows the time to first check-in (measured in weeks). The plot in Figure 6 is a combination of box plot and smoothened density function where the density function indicates the developer distribution pattern. Thus, broader the width; higher the fraction of new hires that takes specific time to make first check-in. In Figure 6, we observe that the percentage increase in the median time to first check-in for middle and senior level software developers is ≈20% relative to entry level software developers. Least median time to first check-in for entry level software developers imply that developers with no or less than a year of prior job experience makes early first check-in compared to experienced new hires. Further, we observe that middle level and senior level software developers have the same median time to first check-in with different density distributions. The density distribution for senior level software developers peaks at median, relative to the spread of middle level software developers. This distribution implies that senior level software developers perform consistently and make early first check-ins compared to middle level software developers. This observation calls for further investigation to understand the factors that influence the time to first check-in for all levels of software developers.

RQ3: Does proximity to core team influence the time to first check-in? Companies adopted distributed development as a strategic response to increasing concerns such as

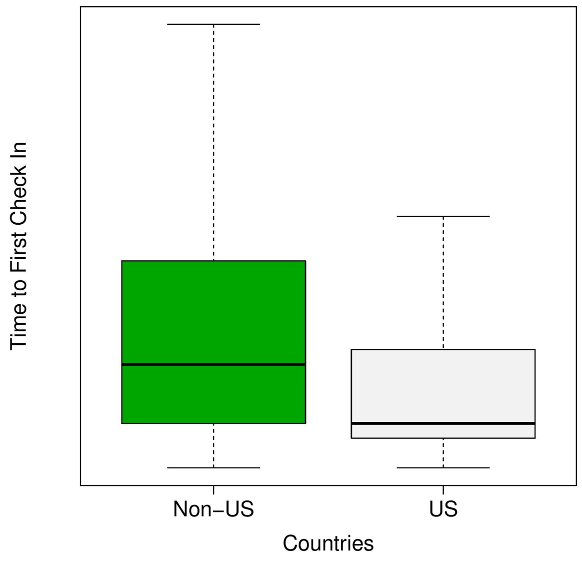


Figure 7: Proximity to core team and first check-in time

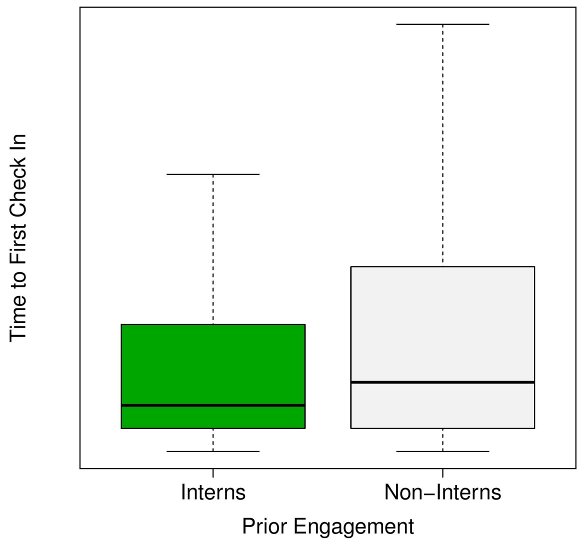


Figure 8: Prior engagement and first check-in time

skill set unavailability, acquisitions, and government restrictions [94]. However, distributed development has its own challenges such as restricted and delayed communication, and less shared project awareness. Owing to these challenges, it is interesting to understand the influence of distributed development on the ramp-up journey of new hires. In particular, here we are interested in studying how proximity to the core of a team in a distributed-development environment influences the time to first-check-in of a new hire.

To study the impact of distributed development, we classified all new hires into two categories: employees hired within United States (US) and employees hired in the rest of the countries (Non-US). In Microsoft, bulk of developers for all product teams is in the US. Therefore, the preceding classification helps us understand the differences in the ramp-up journey when new hires are in proximity to the core team.

We present the box and whisker plot, where the horizontal axis shows the geographical location ('US' and 'Non-US') and vertical axis shows the time (refer Figure 7). In Figure 7, the percentage decrease in the median time to first check-in for new hires in 'US' relative to the new hires in 'Non-US' is $\approx$ 57% when measured in weeks. Thus, new hires in the 'US' category make early first check-in compared to new hires in the 'Non-US' category.

RQ4: Does internship within the company influence the time to first check-in?
Companies make huge investments in internship programs. For example, analysis of different product groups at Microsoft suggested that between 7% and 26% of new hires in those groups had prior internship experience at Microsoft. Apart from giving industry exposure to students, companies view internship programs as a means to identify prospective employees and assess their strengths in real workplace situations [95]. Given its importance, we plan to understand whether new hires with prior internship experience ramp-up faster than others. To study the impact of internships, we classified all new hires into two categories: employees who had prior internship experience at Microsoft (Interns) and others (Non-Interns).

Figures 8 shows the impact of internship on the time to first check-in of new hires. Here, horizontal axis shows the engagement of new hires as 'Interns' vs. 'Non-Interns', whereas, vertical axis shows the time to first check-in. In Figure 8, the median time to first check-in reduces by $\approx$ 33% for 'Interns' relative to 'Non-Interns' when measured in weeks. Thus, new hires who had a prior internship experience at Microsoft tend to make early first check-in compared to other new hires.

2.5.2 *Qualitative Analysis*

To present a comprehensive list of factors that influence the ramp-up journey, we present the opinions of new hires at Microsoft on the factors that influence the time to first check-in. We asked new hires the effect of a set of factors on the time to first check-in. These questions are based on the interviews

Table 3: Influence of the following factors on the time to first check-in [SI: Strong Increase; MI: Moderate Increase; NE: No Effect; MD: Moderate Decrease; SD: Strong Decrease; DK: Don't Know]

How do the following items affect the time to first check-in?	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**Statistical Significance**
Lack of proper documentation for the project	SI	SI	SI	SI	SI	MI	MI	$<0.001^{***}$
Getting access and permissions	MI	MI	MI	SI	SI	MI	MI	$<0.001^{***}$
Working on a code with dependencies to others' work	SI	MI	MI	MI	-	MI	MI	$<0.001^{***}$
Working on preparatory tasks (like code review, coding assignments, etc.)	MI	MI	MI	SI	MI	MI	MI	$<0.001^{***}$
Working on legacy code	MI	MI	NE	MI	MI	MI	MI	$<0.001^{***}$
Availability of resources (like desktop, task related equipment(s) on arrival)	NE	NE	NE	SI	SI	-	MI	0.25
Join the team near product release	NE	NE	NE	SI	DK	NE	DK	$<0.001^{***}$
Changing products, such as moving from Windows	DK	DK	DK	NE	NE	NE	DK	$<0.001^{***}$
Writing new code than fixing issues	NE	NE	NE	MI	MI	NE	DK	$<0.001^{***}$
Identifying the reviewers for the code	NE	NE	NE	MI	NE	NE	NE	$<0.001^{***}$
Changes in team composition, such as change of immediate manager	NE	NE	NE	NE	NE	NE	NE	$<0.001^{***}$
Making check-ins in branches other than the main branch	NE	NE	NE	NE	NE	NE	MI	$<0.001^{***}$

conducted with four new hires with different career stage paths. Table 3 shows the impact of these factors on the time to first check-in. The opinions of new hires are presented on the scale of 'Strong Increase' to 'Strong Decrease'. We use an additional field titled 'I don't know' to account for cases where survey respondents have no opinion on the impact of the factor asked in the survey. Table 3 presents the opinions of new hires in 7 (out of 8) product groups as there was no survey response from one product group analyzed in this study. These observations can be used by the product groups to identify and eliminate bottlenecks in the process and help improve existing practices.

To visualize trends across product groups, we color code the observations in Table 3. Here, we present areas of improvement as shades of red and good practices as shades of green. The factors on which survey respondents have no opinion are presented as yellow. We leave the factors with no effect colorless. Also, '-' indicates that we received insignificant responses from the product division on the influence of the factor to support interpretation. Further, darker the color; stronger is the impact. Thus, dark red (on a relative scale) means that the parameter strongly increases the time to first check-in. In Table 3, all results, except for 'Availability of Resources on Arrival' are statistically significant. We present summaries of statistically significant results and open ended responses.

In Table 3, we see that lack of proper documentation increases the time to first check-in. The lack of proper documentation strongly increases the time to first check-in in 5 (out of 7) product groups. While for the rest 2 product groups, it moderately increases the time to first check-in. Also, getting access and permissions, working on codes with dependencies, and working on legacy codes moderately increases the time to first check-in for majority of product teams. 4 out of 7 product teams say that joining the team near product release has no effect on the time to first check-in, while the other 2 teams did not comment on it. It is noteworthy to see that new hires in product P4 says that joining the team near product release strongly increases the time to first check-in. Besides these, changing product teams, writing new code compared to fixing the issues, identifying the reviewers for the code, changes in team composition, and making check-ins in branches other than the main branch have no effect for the majority of product teams. One key observation from this analysis is that not all product teams are influenced by all the parameters stated above. Also, the degree of influence varies substantially across product teams.

In the open-ended question that followed, we asked survey respondents to enumerate factors, other than the ones listed in the survey, which influence the time to first check-in. We card-sorted the responses in the open-ended question and found the following themes that influence the masses. The themes are presented in the non-increasing order of occurrence.

1. *Mentorship*: Software developers stressed the importance of having a manager, mentor or lead, to talk to, during the initial days. They said that mentors can assist new hires in getting unstuck and make early first check-ins. Also, software developers who were not assigned mentors experienced significant loss of time.

2. *Documentation*: Software developers feel that lack of detailed documentation of products and processes strongly increase the time to first check-in. To add to, the documents are stored at different places, in different formats, and some documentation is out of date.

Table 4: Correlation between time to first check-in and ramp-up time after first check-in

	P1	P2	P3	P4	P5	P6	P7	P8
Commit Counts	-0.06	-0.01	-0.09	-0.22	+0.04	-0.00	-0.04	+0.13
Lines Changed	-0.07	-0.09	-0.24	-0.39	-0.13	+0.10	-0.34	-0.31
Files Changed	-0.13	-0.18	-0.18	-0.52	-0.21	-0.00	-0.16	-0.30

3. *Process*: Software developers feel that engineering processes need some improvement. They believe that the use of standard components, for which documentations and manuals are available widely, will help reduce the time to first check-in.

4. *Access and Permissions*: New hires feel that it takes time to figure out the desired access and permissions. They suggest that it will be helpful to associate access and permissions with the team and not the individuals.

5. *System setup*: Software developers say that they spend a considerable amount of time to set-up environment and configurations, which can be improved.

Besides these, developers find that the lack of confidence to make changes, large size of the code base to study, lack of technical skills required for the job, development environment, meetings with a broader scope (not targeted), and frequent manager changes increase the time to first check-in.

2.6 time to ramp-up

2.6.1 *Quantitative Analysis*

New hires take time to reach the productivity level of existing employees. The amount of time taken to ramp-up influences resource planning, effort estimation, and hence the productivity of the team. Therefore, managers and business analysts might be interested to know the time to ramp-up and study its impact. In this study, we define ramp-up time of new hires as the time required to reach the median productivity level of existing employees. We measure the time to ramp-up on three parameters, namely the frequency of check-ins, lines changed, and files changed. To establish the baseline, we measure the median check-in counts, lines changed, and files changed for existing employees in each product per unit time. We then measure the unit of time it takes for the new hires to reach the median productivity level of existing employees. In this section, we are interested to find answers to the following research questions:

RQ5: Does early first check-in correlates with early ramp-up?
Managers and business analysts might be interested to understand the best practices that help reduce the ramp-up time. We analyze whether early first check-in helps new hires ramp-up faster, compared to others who take longer to make first check-in. For all product teams, we compute the correlation between the time to first check-in and the time to ramp-up after first check-in. We compute the correlation on check-in counts, lines changed, and files changed using Spearman's rank correlation coefficient. For all three parameters, we observe negligible to weak correlation ranging from +0.10 to -0.39 (refer Table 4). Negligible to weak correlation between the time to first check-in and the ramp-up time suggests that the time to ramp-up is not a function of the time to first check-in. However, the negative magnitude of correlation implies that new hires who take longer to make first check-in do not necessarily take longer to ramp-up after first check-in.

RQ6: Is ramp-up journey a function of experience and product?
In the previous section, we examined the impact of experience and product group on the time to first check-in. We observed that the median time to first check-in is similar across all product groups analyzed. Also, we saw that middle level software developers' take longer than entry level and senior level software developers to make first check-in. Here, we are interested to know that similar to the time to first check-in, does experience and product team influence the ramp-up time?

For each product group, we compute the median time to ramp-up and present the results (as shown in Figure 9). In Figure 9, the horizontal axis shows the product groups and the vertical axis shows the time to ramp-up in months. The legend refers to the three parameters we used to measure the time to ramp-up. In Figure 9, we observe that new hires ramp-up on the three parameters stepwise.

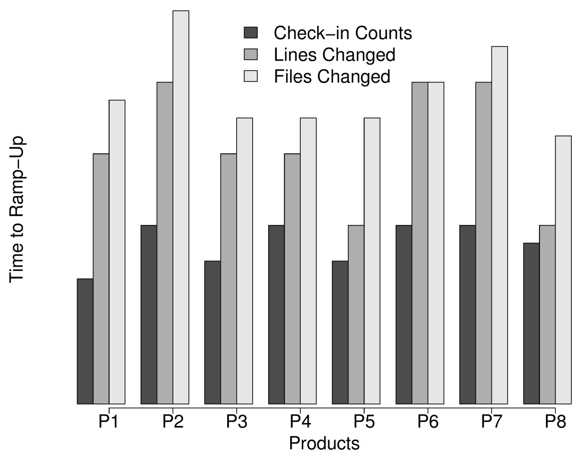


Figure 9: Product groups and time to ramp-up

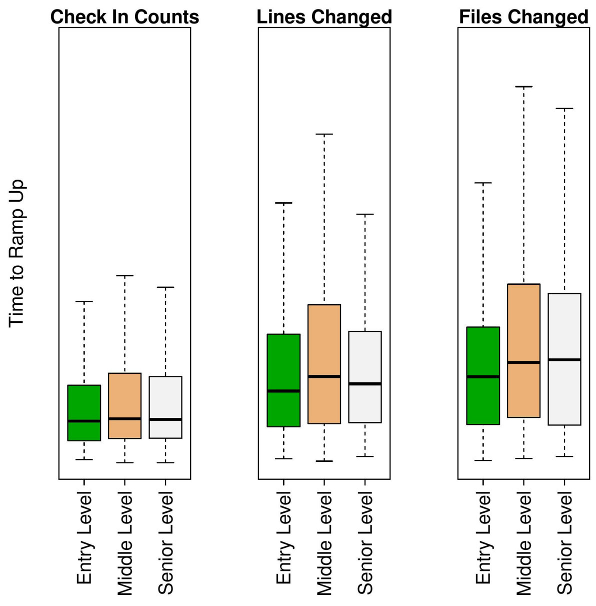


Figure 10: Experience and time to ramp-up

First, new hires ramp-up on check-in counts, followed by lines changed, and files changed. This follows the intuition that making changes to multiple files require broad understanding of the task, and hence takes longer than ramping-up on check-in counts, or lines changed. Also, we observe that the median time to ramp-up on eight product groups is similar for check-in counts (≈32-45% of the maximum time to ramp-up measured in months). However, it varies significantly for lines changed (≈45-81%) and files changed (≈68-100%). A large variance in the ramp-up time on lines changed and files changed indicates differences in the ramp-up time across products. This information can be used by managers and business analysts in understanding the productivity of new hires across products, make better effort estimations, conduct resource planning, and take appropriate actions.

Figure 10 shows the time to ramp-up based on experience. In Figure 10, the horizontal axis shows experience as identified by the title and the vertical axis shows the time to ramp-up (measured in months). We analyze the median time to ramp-up on check-in counts, lines changed, and files changed. We see that the percentage increase in the median time to ramp-up on check-in counts for different experience levels is ≈5%. Thus, experience has no impact on ramp-up time on check-in counts. However, for ramp-up on files changed and lines changed, we see that middle and senior level software developers take marginally longer than entry level software developers to ramp-up. The percentage increase in the median ramp-up time on lines changed for middle level and senior level software developers is ≈13% and ≈6% respectively relative to entry level software developers. Similarly, the percentage increase in the median time to ramp-up on files changed is ≈22% for middle level and senior level software developers relative to the entry level software developers.

RQ7: Does proximity to core team and internship within the company influence the ramp-up time?
In the previous section, we examined the influence of proximity to core team and internship within the company on the time to first check-in. We observed that proximity to core team and internship within the company reduces the time to first check-in. Here, we are interested to know that similar to the time to first check-in, does proximity to the core team and internship within the company influence the time to ramp-up?

Figure 11 shows the ramp-up time of new hires classified by their proximity to the core team. We present the box and whisker plot, where the horizontal axis shows the geographical location ('US' and 'Non-US') and the vertical axis shows the time. In Figure 11, the percentage decrease in the median time to ramp-up for new hires in 'US' relative to new hires in 'Non-US' on check-in counts, lines changed and files changed is ≈8%, ≈6% and ≈4% respectively when measured in months. This implies that the three metrics used to measure the ramp-up time, i.e., check-in counts, lines of code changed, and the number of files changed show only a marginal difference in the median ramp-up time among new hires in 'US' and 'Non-US'. New hires in 'US' take marginally less time than their 'Non-US' counterparts. Our observations suggest that proximity to the core team does help the

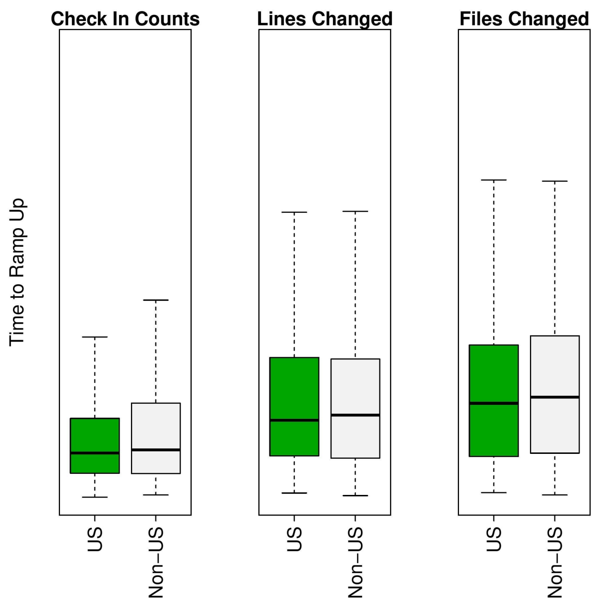


Figure 11: Proximity to core team and time to ramp-up

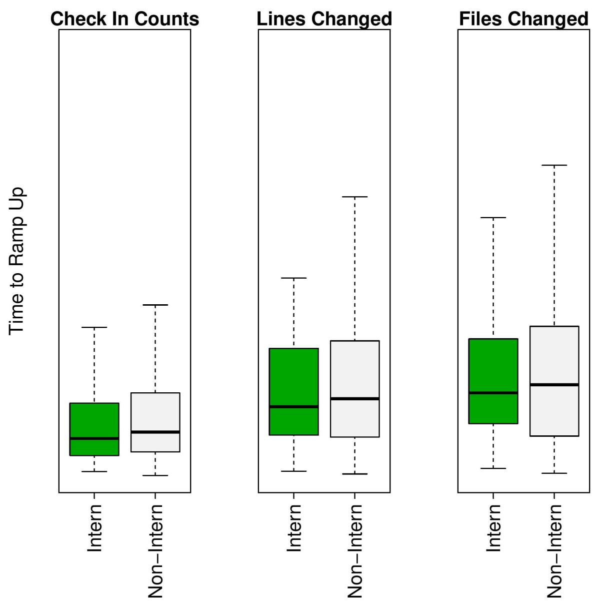


Figure 12: Prior engagement and time to ramp-up

ramp-up journey of new hires in the beginning, however, the effect decreases in the overall ramp-up journey.

Figure 12 shows the impact of internship on the ramp-up journey of new hires. Here, the horizontal axis shows the engagement of new hires as 'Interns' vs. 'Non-Interns' whereas the vertical axis shows the ramp-up time. In Figure 12, the percentage decrease in the time to ramp-up for 'Interns' relative to 'Non-Interns' on check-in counts, lines changed and files changed is ≈10%, ≈6%, and ≈5% respectively. Thus, interns ramp-up faster compared to non-interns, however, the time difference is marginal compared to the time to first check-in (refer Figures 8 and 12). We also observe similar patterns for other quartiles. Therefore, our observations indicate that familiarity with people, process, and product acquired during the internship experience indeed influence the time to first check-in. However, the effect attenuates in the long run as non-interns only take marginally more time to ramp-up than their counterparts. This observation implies that among other factors, internship experience does help new hires to ramp-up faster.

2.6.2 *Qualitative Analysis*

We present the opinions of new hires at Microsoft on the factors that influence the time to ramp-up. We asked the new hires the effect of the factors (mentioned in Table 5) on the time to ramp-up. These questions are based on the interviews of new hires who followed different career stage paths. Table 5 shows the central tendency summaries, as presented by mode, for the 7 (out of 8) product groups analyzed in the study. Similar to the previous section on qualitative results, here red indicates areas of improvement, green indicates good practices, yellow implies don't know, and colorless means no effect. Also, darker the color means stronger the impact. In Table 5, all results, except for 'Communicating Technical Prerequisites', are statistically significant.

In Table 5, we observe that prior knowledge of programming languages, programming environment, and tools help decrease the ramp-up time. Similarly, proactively asking questions, prior familiarity with the process, and having a mentor decrease the time to ramp-up. Though, the influence is different for products P4 and P5. Contrary to this, new hires say that maintaining documentations, to do lists, and working on preparatory tasks increase the ramp-up. When asked about the impact of prior familiarity with the team on ramp-up time, new hires expressed mixed opinions across product teams. Also, 5 (out of 7) teams say that active participation in social events has no effect on the time to ramp-up. The rest 2 teams say that it moderately increases the time to ramp-up. In addition to the parameters stated above, software developers feel that the following practices influence their time to ramp-up:

1. *Team Interaction*: New hires say that verbal communications in team and pair programming are the most effective ways to ramp-up. They find that spending more time with the manager and

Table 5: Influence of the following factors on the time to ramp-up [SI: Strong Increase; MI: Moderate Increase; NE: No Effect; MD: Moderate Decrease; SD: Strong Decrease; DK: Don't Know]

How do the following items affect the time to ramp-up?	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**Statistical Significance**
Prior knowledge about programming languages (such as C#), programming environments (such as Visual Studio), or tools (such as versioning tools)	SD	MD	SD	-	MD	-	SD	$<0.001^{***}$
Proactively asking questions to your manager, mentor or team	SD	MD	MD	-	-	SD	MD	$<0.001^{***}$
Prior familiarity with processes (such as how effort estimation is done or review process)	MD	MD	MD	MI	MI	MD	MD	$<0.05^{**}$
Having a mentor	SD	SD	SD	SI	SI	SD	DK	$<0.001^{***}$
Maintaining documentation, to do lists, introductory videos for different employee titles	SI	NE	SD	SI	MI	MI	MI	$<0.001^{***}$
Preparatory tasks (such as code review, building prototypes) or training programs (like boot camp)	MI	MI	MD	SI	SI	MI	SI	$<0.001^{***}$
Communicating the relevant technical prerequisites (such as tools and languages used) of an employee's title, prior to joining	MD	NE	NE	MI	SI	MI	MI	0.66
Prior familiarity with team (such as the cases of moving from an intern to a full time position)	NE	DK	DK	MI	MI	DK	DK	$<0.001^{***}$
Active participation in social events, such as team lunches etc.	NE	NE	NE	MI	MI	NE	NE	$<0.001^{***}$

the team during first 1-2 months is helpful. They add that recently ramped-up employees reduce the ramp-up time of new employees the most.

2. *Training*: Software developers say that training programs like boot camp, etc. are very helpful.
3. *Overview of the system*: Software developers say that well-chosen starting tasks that gives a complete overview of the system helps reduce the ramp-up time.
4. *Proximity to release*: Developers find that joining the team after product release increases the time to ramp-up as there is not much code to write.

Other than these, developers' say that familiarity with people, customer, process, product, and code helps improve the ramp-up journey.

2.7 OTHER NEW HIRE ACTIVITIES

In the above two sections, we presented different factors that influence the ramp-up journey and measured time to ramp-up using various features of check-ins. However, new hires perform various activities, other than code check-ins, thereby making it important to understand the activities that are not captured in the study. We asked the survey respondents the list of activities they perform and present the themes arranged as different stages in the ramp-up journey of new hires.

New hires who relocated from other countries or states say that it takes time to settle in a new place. Further, they add that on arrival, they have to set-up the system, get the required access and permissions, enroll for benefits, get HR/staffing set up, etc. Once the required resources are made available, new hires find themselves trying to understand the existing system and identify their role in the team. They say that they undergo training programs, attend meetings, and study all kinds of documentation to acquire the required technical and functional knowledge. They add, that sometimes they are assigned code reviews, prepare prototypes or demos, or even read legacy code to get a better understanding of the system. Further, in the process of knowledge transfer, they spend substantial time in proactively asking questions. New hires emphasize that while they are learning and trying to ramp-up, they also engage in other activities like testing, bug fixing, debugging, bug triaging, identifying and resolving dependencies, etc.

New hires also identify some miscellaneous activities that claim their time and effort. They say that they spend time in planning, writing proposals, estimate time for the task, perform production related duties, add work items, participate in events like 'Hackathon', etc. While this list is not exhaustive, it gives a fair understanding of the various activities that new hires perform other than code check-ins. This list of activities suggest that while code check-in is a good indicator of productivity, it does not present a complete picture.

We requested new hires for suggestions on practices that might help improve the productivity of new hires. We believe that this information can help companies in improving the ramp-up journey of new hires. New hires suggest that improvements in engineering systems like applying companywide coding standards, improved code base and documentation, easy tools, etc. will increase the productivity of new hires. These findings align with Microsoft's recent initiatives such as One Engineering System to have one common system across all projects. They emphasized the usefulness of training tools and sessions, and guided work for a few weeks during ramp-up time. They also added that centralization of all information and clearly communicated expectations are other factors that accelerate the ramp-up journey.

2.8 THREATS TO VALIDITY

2.8.1 *Internal Validity*

- *Data accuracy:* The accuracy of the results of this study depends on the accuracy of the data on which it is built, e.g., some data may be missing or incomplete. We believe that this is only a minor threat. For the study, we used the CodeMine tool, which attempts to capture software development activities as accurately and completely as possible. Several production systems at Microsoft are built on top of CodeMine and its accuracy has been extensively verified.

2.8.2 *Construct Validity*

- *Activities in other product groups:* We analyze commits in the eight product groups, which constitute a vast majority of the Microsoft workforce. However, if developers engage in activities in product groups other than the ones analyzed here, we are not able to capture their contribution.
- *Activities other than code check-ins:* We compute the ramp-up journey of new hires in terms of code check-ins. However, new hires engage in a wide variety of activities other than code check-ins (refer Section 2.7). Also, different product groups emphasize on different set of activities during the ramp-up journey of new hires. These two factors may influence the observed time to first check-in and the time to ramp-up. So, while comparing product groups on the time to first check-in and the time to ramp-up, these two factors should be taken into consideration.

2.8.3 *External Validity*

- *Application of results to product divisions within and outside Microsoft:* We analyzed eight large, popular product teams, which constitute the majority of Microsoft's engineering workforce. We, therefore, believe that the results are widely applicable to product divisions in Microsoft. We do not claim that the findings and recommendations presented in this study extend to any organization and product team. For example, we expect that the findings may not generalize to organizations with bootstrap mechanism different from Microsoft or organizations that hire software developers for testing purposes only. While findings may not generalize, the research methodology can be applied to other contexts as long as there are sufficient data points to compute a baseline productivity of existing employees.
- *Geographic differences:* The survey included participants from different countries with different cultures and working hours. In addition, Microsoft relocates new hires from other countries to new countries. We did not collect enough data to analyze and control for this effect. We, therefore, caution the readers that our findings may not apply to arbitrary countries and cultures.

2.9 SUMMARY

In this chapter, we described quantitative and qualitative analysis to understand the factors that influence the ramp-up journey of new hires. The results of this study reiterate some of the knowledge already known to the industrial world by mining software engineering data. We analyzed eight large product groups at Microsoft and observed that the time to first check-in, a milestone in the ramp-up journey of new hires, is invariant to the product group analyzed. In terms of experience, as indicated by the career stage path levels, entry level software developers make faster check-ins compared to middle and senior level software developers. Proximity to the core team and internship within the company reduces the time to first check-in. To complete the analysis, we asked the opinions of new hires to understand the factors that influence the time to first check-in. We observed that among other factors, lack of proper documentation, getting access and permissions, etc. increase the time to first check-in. Further, we computed the ramp-up time of newly hired software developers on check-in counts, lines changed, and files changed. We observed that first new hires ramp-up on check-in counts, followed by lines changed, and files changed. We also found that the time to first check-in is weakly, if at all, correlated with the ramp-up time. The negative correlation implies that new hires who take longer to make first check-in do not necessarily take longer to ramp-up thereafter. We see that ramp-up time is a function of experience and product on lines changed and files changed. Also, proximity to the core team and internship within the company has a marginal effect on the ramp-up time of new hires. In addition to this, survey results suggest that prior knowledge of required technical skills, proactively asking questions, and familiarity with the process help reduce the ramp-up time along with other factors. We also list activities, other than the code check-in, that claim developers' time and efforts. We conclude the study with suggestions of new hires to help improve the productivity of new hires.

3 GEOGRAPHICAL BIAS IN PEER CODE REVIEW

Open source development has often been considered to be a level playing field for all developers. But there has been little work to investigate if bias plays any role in getting contributions accepted and developers' perceptions of the bias. In this chapter, we present a study - one of the largest of its type, to understand the perceptions and reality of the influence of geographical location on the evaluation of pull requests in GitHub - one of the primary open source development platforms. Using a mixed-methods approach that analyzes 70,000+ pull requests and 2,500+ survey responses, we found a bias blind spot.

Data analyses showed that geographical location explains differences in pull request acceptance decisions. Compared to the United States, submitters from the United Kingdom(22%), Canada(25%), Japan(40%), Netherlands(43%), and Switzerland(58%) have higher chances of getting their pull requests accepted. However, submitters from Germany(15%), Brazil(17%), China(24%), and Italy(19%) have lower chances of getting their pull requests accepted. The probability of pull request acceptance increases by 19% when the submitter and integrator are from the same geographical location. Survey responses from submitters indicate that the perceptions of bias are strong in Brazil and Italy, matching the data analysis. However, integrators do not perceive as being biased.

3.1 INTRODUCTION

Biases have been found to hurt meritocracy in offline work groups [96][97]. For years, visible characteristics have been used to differentiate people in all spheres of life ranging from sports [98] to health care [99] to job applications [100]. The perceived differences in values and norms point towards the likelihood of engaging in stereotyping, cliquishness, and conflicts [101]. Recently biases have been reported in online environments too [102][103].

Open source software (OSS) development started as a merit-based model [104] which gave rise to terms like 'code is king' [104][105]. Social factors were found to influence work-related decisions [106], in addition to the technical factors. In recent years, social work environments like GitHub [107], Bitbucket [108], etc. have gathered a large number of developers. These platforms provide transparency and access to developers' profiles. The increasing level of awareness of demographic attributes of fellow contributors makes it important to understand the reaction of the community to this diversity.

Studies on GitHub have looked into the influence of visible demographic attributes like gender, tenure, etc. on the presence of bias [65] and productivity of teams [64]. Our goal with this work is to understand whether the geographical location of developers influences the way their contributions are evaluated. We choose to study the influence of geographical location on the evaluation of contributions for the following reasons: 1) its observed impact on work-related decisions in offline groups [96][102], and 2) a reasonable degree of visibility of geographical location in social work environments [109]. Through this study, we intend to generate awareness of the presence of bias and bridge gaps in perceptions, if any. Studies show that individuals can only work to correct for sources of bias that they are aware exist[1] [110]. Awareness may prompt individuals to pursue corrective measures [111]. In terms of Nathaniel Branden [112]

"The first step towards change is awareness..."

To examine bias in online, distributed software development, we leverage GitHub - the largest, most popular online collaborative coding platform. We study the pull-based development model - one of the most popular models for collaborative development (45% of collaboratively developed repositories use the pull-based development model), which has all the characteristics of an online and distributed development environment.

One of the key challenges in conducting this study is to detect the presence of bias when the developers themselves might not be aware of it. Even when developers are aware of their biases, it is hard to make developers accept them. For this reason, we use a mixed-methods approach. We

1 http://www.ncsc.org/~/media/Files/PDF/Topics/Gender%20and%20Racial%20Fairness/IB_Strategies_033012.ashx

combine observations from 70,000+ pull requests and 2,500+ survey responses - one of the largest response population of open source projects[113], to analyse the influence of geographical location on pull request acceptance decisions.

We quantitatively analyse GitHub projects' data to measure the influence of 1) the geographical location of submitters on pull request acceptance decisions and 2) the same geographical location of submitters and integrators on pull request acceptance decisions. We examined pull request acceptance rate across geographical locations as a proxy to geographical bias. We support these observations with the quantitative and qualitative analysis of the survey responses of submitters and integrators on the perceptions of bias. The two roles of developers: submitter and integrator, together present the main stakeholders. We combine results to understand the difference (or agreement) between actual and perceived bias on geographical location.

We find a bias blind spot - a type of cognitive bias where integrators perceive the absence of bias while submitters experience it - similar to the results from data analysis. This study informs both integrators and submitters about the actual presence of bias despite their reported perceptions.

3.2 BACKGROUND AND RELATED WORK

3.2.1 *Visible Demographic Attributes and Bias*

The relationship between visible demographic attributes (like race, gender, etc.) of people and work-related decisions in offline groups had been a subject of study for years [99][100][114]. The influence of these diversities was felt in online communities too. The reaction of the community to these diversities depends on the extent to which these features are salient [64]. OSS started as a merit-based model [115], however, with the rise of social work environments, like GitHub, developers are somewhat aware of the demographic features (age, gender, ethnicity, etc.) of the fellow developers [109]. This awareness has been used to form impressions using the history of activities [116]. This is also analysed to understand the influence of gender and tenure diversity on team productivity [64] and the presence of gender bias [65] in GitHub.

3.2.2 *Pull-based Development*

GitHub supports two models of collaboration: the shared repository model and the pull-based development model. The pull-based development model separates development efforts from the decisions to include the submitted code [117]. This separation allows projects to be more democratic and transparent, which has increased participation [118]. Currently, less than half of the collaboratively developed projects exclusively or complementarily use this model [117].

3.2.3 *Factors Influencing Pull Request Acceptance Decisions*

Factors influencing pull request acceptance decisions can be broadly classified as developer characteristics, project characteristics, and pull request characteristics. For a developer, reputation (technical and social) is seen to positively influence pull request acceptance decisions [106][119][120][121]. Following technical and social norms are seen to increase the chances of contribution acceptance [106][116][120]. For a project, maturity and popularity are related to lesser chances of pull request acceptance [106][120]. Also, the nature of the pull request, measured as its size (source code churn), quality (including test cases) and uncertainty associated with it (amount of discussion) influence the chances of contribution acceptance [106][120][122].

3.3 METHODOLOGY

We use a mixed-methods approach [19][16] and triangulate our observations by combining GitHub projects' data with survey response data. First, we carefully select a dataset of GitHub developers and projects and model the influence of geographical location on the pull request acceptance decision by controlling for confounding effects. Further, we conduct two large-scale surveys of submitters and integrators in GitHub and quantify their perceptions and experiences. Conducting surveys of submitters and integrators helped us analyze differences (or agreement) between the actual and perceived

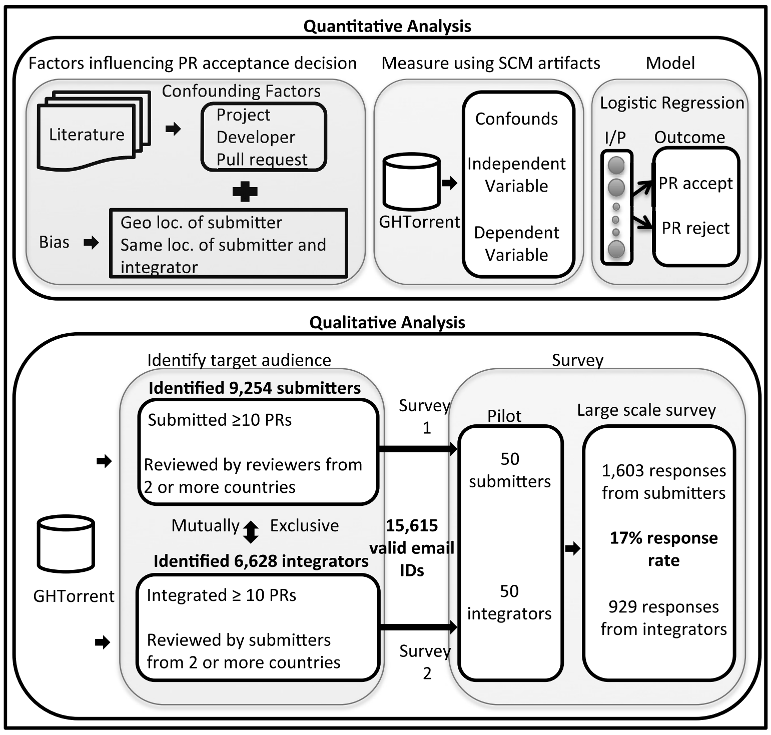


Figure 13: Research Method: Mixed-Methods Approach

bias on geographical location. Below, we present a detailed description of our selection procedure and analysis methods. A diagrammatic presentation of the research method is shown in Figure 13. All the data and procedures used in this section are made publicly available for replication [123]. We strongly encourage researchers to download the data and build on our work.

3.3.1 *GitHub Data*

3.3.1.1 *Feature Selection*

Factors influencing pull request acceptance decisions are borrowed from the literature in software engineering [106][64][120] and social sciences [99][100][114][124]. Table 6 presents a comprehensive list of factors that are seen to influence pull request acceptance decisions. In addition to these factors, for our study, we borrow the concept of bias based on the geographical location from the social sciences literature [125] and measure it in terms of software engineering data and its associated meta-data. The list of features used in this study is presented in the last column of Table 6.

3.3.1.2 *Data Collection*

We downloaded the GitHub projects' data designed to analyse the pull request development model, made publicly available by Gousios et al. [113]. We enrich the dataset with additional information required for this study from the GHTorrent dataset made available on August 18, 2015 [126]. We also use the GHTorrent dataset for surveys, as we discuss later. The enriched dataset is a collection of 1,069 projects and 370,411 pull requests developed in Python (357), Java (315), Ruby (359), and Scala (38). These carefully selected projects represent top 1% of the projects developed by using the pull request development model. It combines GHTorrent data with project repositories data and provides a list of features seen to influence pull request development. For this study, we augment this dataset with pull request life cycle information, participants' demographic information, and measures of social norms that influence pull request acceptance decision. A description of the procedure to extract the above-mentioned factors from the GHTorrent dataset follows:

3.3.1.3 *Features*

PARTICIPANTS' DEMOGRAPHICS We measure the geographical location of developers as the location (or country of residence) specified by developers in their GitHub profiles. Following two factors

Table 6: A comprehensive list of factors influencing pull request acceptance decisions

Characteristics		Measure			
		Tsay et. al [106]	Gousios et. al [120]	Current study	Representation
Project characteristics					
Maturity		Project tenure	-	Project tenure	repo_pr_tenure_mnth
Team size		Count of collaborators	Active core team	Active core team	-
Popularity		Watchers' count	Watchers' count	Watchers' count	repo_pr_popularity
Size of code		sloc	sloc	sloc	sloc
Openness		-	% of external contribution	% of external contribution	perc_external_contribs
		-	Test lines per kloc	Test lines per lloc	test_lines_per_lloc
Test based code quality		-	Test cases per kloc	Test cases per lloc	-
		-	Asserts per kloc	Asserts per lloc	-
Developer's *acquired characteristics*					
Social skills	Status in community	Followers	Followers	Followers	prs_popularity
	Status in project	Project membership	Project membership	Project membership	prs_main_team_member
Social norms	Follow the integrator a priori	Follow the integrator prior to contribution	-	Follow the integrator prior to contribution	prs_followed_pri
	Follow the repository a priori	Follow the repository prior to contribution	-	Follow the repository prior to contribution	prs_watched_repo
Technical skills	Experience/Expertise	-	Previous pull requests	Previous pull requests	prev_pullreqs
		-	Submitter success rate	Submitter success rate	prs_succ_rate
		Submitter tenure	-	Submitter tenure	prs_tenure
Technical norms	Size of change	src churn	src churn	src churn	src_churn
		Files changed	Files changed	Files changed	files_changed
	Test cases	Test inclusion	Test churn	Test inclusion	test_inclusion
Developer's *innate characteristics*					
Geographical location		-	-	Country of residence	prs_location
Explicit or implicit bias on geographical location		-	-	Likelihood of PR acceptance on geo loc	measured using prs_location
In-group bias		-	-	Likelihood of PR acceptance on same geo loc	measured using prs_pri_same_location
Pull request characteristics					
Uncertainty associated with pull requests		Comments count	Comments count	Comments count	num_issue_comments

motivates our choice of country of residence as a measure of geographical location. First, the perceptions of fellow developers around geographical location are framed in terms of the location specified in GitHub profile. Second, we believe that work habits and cultural environment, specific to the location of current residence of a developer, may explain differences in the behaviour. One may argue that these differences are caused by a combination of current and attenuating effects of past locations of residence. However, for simplicity, we study the differences in pull request acceptance decisions in terms of current location only.

In GitHub, mentioning the location is optional and is specified in a free-form text. Thus, developers can write 'US', 'United States', 'XYZ Apartments, New York', etc. to refer to the same location. To identify the geographical location of developers irrespective of the format of the location, we use 'countryNameManager' script used by Vasilescu et. al [64]. This script uses free-form text to identify the geographical location of GitHub users, who chose to disclose it. Further, to identify the geographical location of developers who did not mention it explicitly, we proposed some heuristics.

The proposed heuristic uses the data points for which geographical location is identified using 'countryNameManager' as training data. It then uses domain names of email addresses and affiliation of developers to identify their geographical location. The underlying principle used by the heuristics is that the affiliation and domain name can be used to localise country of residence. For instance, affiliation to 'Peking University' maps the country of residence to China. However, this approach is prone to false-positives. To minimise false positives, we choose one-to-one mapping between predictors and geographical location (exclude one-to-many) and set the threshold for inclusion to 20. Thus, if in our training dataset the company name 'Peking University' maps to China only and has at least 20 data points to support it, we map 'Peking University' to China. This information is used to identify the geographical location of developers for whom we cannot identify the geographical location using

Table 7: Pull request data generated

ID	State: Open	State: Merge	State: Close	Status	Same Loc
1	Dev1 (India)	-	-	Open	-
2	Dev1 (India)	Dev2 (India)	-	Merged	Yes
3	Dev1 (India)	Dev3 (US)	Dev3 (US)	Merged	No
4	Dev1 (India)	-	Dev4 (France)	Not Merged	No
5	Dev1 (India)	Dev1 (India)	Dev1 (India)	Merged	Self

'countryNameManager' script and who have mentioned company name or email address. This way, we identified the geographical location of 149,268 developers (out of 541,685) who participated in pull request based development. Here, participants refer to both submitters and integrators.

LIFE CYCLE OF A PULL REQUEST A pull request is opened by a submitter and is in state 'open'. Integrators (core team developers) review the pull request. The integrators evaluate the opened pull request and decide to 1) merge the suggested change in the original code with a state 'merge', followed by state 'close' or 2) close it directly without merging it with a state 'close'. A pull request can be in state 'open', 'merged', or 'not merged'. A pull request may get re-opened multiple times. However, here for simplicity, we focus on the pull request lifecycle starting from the time when the pull request was opened the first time until it gets closed the first time.

We select pull requests which are 'merged' and 'not merged', and exclude 'open' pull requests. We construct the pull request life cycle and append it with the developer of the action, geographical location of developer and other attributes. Here, developers who opened pull requests are marked as 'submitter'. Developers who merged the pull requests are termed as 'merger' and developers who closed the pull requests as 'closer'. Together, 'merger' and 'closer' are integrators and are the core team of the project. The final data from this exercise looks like the one in Table 7. In Table 7, submitter 'Dev1' from India interacts with integrators 'Dev1', 'Dev2', 'Dev3', and 'Dev4' from various geographical locations to get her pull requests reviewed. The outcome of the pull request review (status) and the relationship between the geographical location of submitters and integrators (same geographical location) is identified.

3.3.1.4 *Pull Request Project Sample*

The geographical location of submitters follows a highly skewed distribution (kurtosis: γ=98.2). So, to ensure that we have diverse geographical locations and significant pull request counts for each location, we select geographical locations, which represent at least 1% of the total pull requests of the GitHub data. Thus, the United States (38%), the United Kingdom (8%), Germany (6%), France (5%), Canada (4%), Japan (3%), Brazil (3%), Australia (2%), Russia (2%), Netherlands (2%), China (2%), Spain (2%), India (2%), Switzerland (1%), Sweden (1%), Italy (1%), and Belgium (1%) with at least 1% of total pull requests are selected for analysis. These selected 17 geographical locations represent approximately 83% of the total developer population for whom we were able to identify the geographical location. We observed that the submitters themselves merge a significant fraction of pull requests on GitHub. An analysis of 113,191 pull requests in the enriched dataset showed that other developers integrate 63% of the pull requests and the submitters themselves integrate the remaining 37%. Similar statistics are observed in the GHTorrent dataset (44% merged by self, 56% by others). So, by selecting pull requests from submitters from the selected 17 geographical locations, eliminating pull requests merged by self and incomplete data, we are left with 70,740 pull requests for analysis.

3.3.1.5 *Statistical Methods*

In this study, a pull request is the base unit of analysis. Each pull request is an independent observation characterized by temporally evolving project, developer and pull request characteristics. To understand the influence of geographical location of a submitter and its interaction with the geographical location of an integrator on the pull request acceptance decision, we test two hypotheses.

$H1_0$ There are no differences in the pull request acceptance decisions based on the geographical location of submitters.

$H1_a$ There are differences in the pull request acceptance decisions based on the geographical location of submitters.

$H2_0$ There are no differences in the pull request acceptance decisions based on the same geographical location of submitters and integrators.
$H2_a$ There are differences in the pull request acceptance decisions based on the same geographical location of submitters and integrators.

To test the two hypotheses, we use regression modeling. We model the pull request acceptance decision as a binary classification problem. Specifically, we use logistic regression, as implemented in R [127][128]. We measure statistical significance at a p-value <=0.05, the size of change as log odds, and the impact as the percentage of deviance as used in other studies [4]. Though this measure provides an interpretation similar to the percentage of the total variance explained by least square regressions, the two measures are not the same [129].

At a finer level of details, we compute a pairwise correlation of continuous variables and note down the two highly correlated variables. We consider two variables as highly correlated when their correlation coefficient is greater than 0.7 [130]. Similarly, to measure the relationship between two categorical variables, we use a chi-square test for dependence for significance and measure its effect size using Cramer's V [131]. For categorical variables, we consider that the relationship between two variables is strong if it exceeds 0.7 [132]. We identify and note strongly correlated categorical variables too. We model the relationship between a set of predictors and the response variable. Next, to stabilise the variance, we log-transform the independent count variables. We verify this by using AIC and Vuong test for non-nested models [133] to compare the transformed and non-transformed data. To check that multicollinearity is not an issue, we compute the Variance Inflation Factor (VIF). Any VIF value greater than 5 is considered to indicate multicollinearity, as used in various studies [129]. At this step, we eliminate highly correlated variables that cause multicollinearity. We measure the fitness of the model using Area under Curve (AUC). The value of AUC should be greater than 0.5 for the model to be acceptable[2].

Once the model is built, we read the coefficients of logistic regression as the expected changes in the log of responses for a unit change in the predictor variable while keeping all other predictor variables at a constant value. So, for continuous predictor variables, one unit change in its value is associated with an exponent of the coefficient change in the response variable. The interpretation is slightly different for categorical variables. To analyse the effect size of statistically significant features, we use dummy or treatment code to compare a base level with treatments. Specifically, to measure the influence of geographical location, we choose the United States, with the majority of developers, as the base level. Similarly, to test for bias based on the interactions between the geographical location of submitters and integrators, we choose 'different geographical location' as the base level.

3.3.2 *Survey*

We designed two surveys - one from the perspective of submitters and the other from the perspective of integrators. The choice of conducting two separate surveys helps us understand perspectives and experiences.

3.3.2.1 *Design*

Since the goal of conducting two surveys is to learn from the developer community rather than a few individuals, we designed online surveys [134] [135]. We did not give any monetary incentives. The survey was designed to take a maximum of 7 minutes for survey respondents and was active for three weeks.

There were two rounds for each survey. First, we identify 50 developers each for the role of the submitter and the integrator and sent them the survey as part of the pilot study. Based on their feedback and refinement, we sent the main survey to all submitters and integrators identified to

2 https://www.kaggle.com/wiki/AreaUnderCurve

answer our surveys. We explicitly informed our prospective survey respondents that the results of the study would be anonymous to ensure that developers share their true opinion and not try to present themselves in good light. The details of each survey are given below:

integrators We asked our survey respondents four types of questions. First, we asked them a few demographic questions to get an understanding of the diversity and representativeness of the survey responses. This was followed by questions to understand their perspectives on the presence of bias. These questions are:

1. The level of awareness of the demographic features of the submitters they work with.
2. Perceived relevance of the importance of developers' characteristics.
3. Explicit perceptions of bias based on the geographical location of submitters and its interaction with the geographical location of integrators.

In total, we asked 12 questions, out of which there were 8 multiple-choice questions, 3 Likert scale questions and 1 open-ended question. The aim of the open-ended question was to discover factors other than the ones mentioned in the survey.

submitters We asked our survey respondents two types of questions. Similar to the survey for integrators, we started with asking a few demographic questions. This was followed by questions to understand the following:

1. Their understanding of the importance of factors influencing pull request acceptance decision of integrators.
2. Their personal experiences with bias.

In total, we asked them 11 questions, out of which there were 8 multiple choice questions, 2 Likert scale questions and 1 open-ended question.

Throughout the two surveys, we used 5-point Likert scale for the study with one exception. To understand the perceptions of submitters on "The important of factors influencing pull request acceptance decision of integrators", we provided an additional choice - 'I don't know'. This choice was provided to account for cases when submitter doesn't know the perceived importance of the factor.

3.3.2.2 *Identify Survey Respondents*

From the GHTorrent dataset, we extracted developers' geographical location information, role, and contribution count to identify a list of survey respondents for this study. We selected all submitters who have submitted at least 10 pull requests that were reviewed by integrators from various geographical locations (at least 2). Here, the count of pull requests submitted does not include the pull requests closed by self. Similarly, we identified integrators who reviewed at least 10 requests of submitters from various geographical location (at least 2). Here also the count of the pull requests excluded the pull requests merged by self. These selection criteria ensure the following:

1. *Candidacy of the developer to answer our survey questions*: The choice of two or more geographical locations ensures that the developer has experience working with developers from diverse geographical locations. Such developers can help us understand the influence of geographical location on the pull request acceptance decisions.
2. *Reasonable experience and diversity of respondents*: The choice of working on at least 10 pull requests not closed by self, ensure that the respondent has reasonable experience working with the pull-based development model. It ensures that we include a wide range of developers for the study.

Developers can play the roles of submitter and integrator simultaneously. To select unique respondents per survey, we select developers for the role they worked the most. So, a developer who wrote 100 pull requests and reviewed 150 pull requests is considered for the role of the integrator. Following this approach, we identified 6,628 integrators and 9,254 submitters. Out of these 15,882 prospective respondents, only 15,615 respondents had a valid email address.

We sent customised, personally addressed emails to all identified prospective survey respondents. We informed the developers about their contribution in terms of the approximate count of pull requests they worked on and an approximate count of the geographical locations with which they collaborated. This customised report generated interest in the survey, which was reflected in the 500+ email responses received and 2,532 survey responses - one of the largest survey responses with 17% response rate (excluding 818 messages which failed to deliver).

3.3.2.3 *Data Analysis*

We identified all complete survey responses, preprocessed it and got it into a form usable for analysis. To hypothesis test the research questions, we converted the ordinal data to its nominal equivalent and conducted chi-square test.

We started with some basic understanding of the diversity of responses in terms of respondents' demographics. This is followed by the perceived importance of the geographical location of the submitter on the pull request acceptance decision across the two roles. We additionally analysed the level of awareness of the integrators regarding submitters' demographics. Finally, to understand the perceptions of bias based on geographical location, we asked submitters and integrators different questions.

- *Question for submitters* Did you experience bias based on your geographical location in getting your pull requests accepted?
- *Questions for integrators* How much do you agree with the following statement?
  - It is easy to work with developers from the same geographical location.
  - I encourage developers from my geographical location to contribute.
  - Developers from some geographical locations are better at writing pull requests relative to others.

To understand the perceptions of developers, we hypothesis tested the response as follows

H_0 The experiences on bias are of equal proportions.
H_a There are unequal proportions of experiences on the perceptions on bias.

If at a 0.05 significance level null hypothesis is rejected, we measure the effect size as the percentage difference. Next, we see if there are significant differences in perceptions of bias across geographical locations. If the differences are significant, we measure the differences in the experiences around bias using logistic regression. For this, we select geographical locations for which we received at least 10 responses.

We codified open-ended survey response to get an understanding of the justification of integrators around the presence of bias based on geographical location. The first author started with identifying themes by analysing first 100 comments. The themes were the top three suggestions that integrator would like to give to submitters to improve the chances of their pull requests getting accepted. These themes, along with any other theme that evolved in time were codified for all open-ended responses. One other author then verified this. These suggestions conveyed the expectations of integrators from submitters and also pointed towards the possible explanation of observed differences in perceptions.

3.4 RESULTS

3.4.1 *Analysis of GitHub projects' data*

To answer our research question, we test Hypothesis 1 and Hypothesis 2 using GitHub projects' data. We model the influence of geographical location on pull request acceptance decisions using logistic regression while controlling for the effect of confounding factors (refer Table 8).

H1: There are differences in the pull request acceptance decisions based on the geographical location of submitters.

In Model 1 of Table 8, we see that increase in maturity, popularity, the size of the code, and openness to external contribution reduces the chances of the pull request acceptance decision while a well-tested code - an indicator of code quality increases it. We see that increase in technical skills of

Table 8: Logistic regression model of factors influencing pull request acceptance decision [AUC: 0.7]

	Model 1	Model 2
(Intercept)	$2.82\ (0.14)^{***}$	$2.61\ (0.14)^{***}$
Control variables		
repo_pr_tenure_mnth	$-0.01\ (0.00)^{***}$	$-0.01\ (0.00)^{***}$
repo_pr_popularity	$-0.00\ (0.00)^{***}$	$-0.00\ (0.00)^{***}$
perc_external_contribs	$-0.01\ (0.00)^{***}$	$-0.01\ (0.00)^{***}$
test_lines_per_lloc	$0.00\ (0.00)^{***}$	$0.00\ (0.00)^{***}$
log(sloc + 1)	$-0.06\ (0.01)^{***}$	$-0.06\ (0.01)^{***}$
prs_tenure	$-0.00\ (0.00)^{***}$	$-0.00\ (0.00)^{***}$
log(prev_pullreqs + 1)	$0.17\ (0.01)^{***}$	$0.17\ (0.01)^{***}$
prs_succ_rate	$0.01\ (0.00)^{***}$	$0.01\ (0.00)^{***}$
test_inclusion1	$0.26\ (0.03)^{***}$	$0.26\ (0.03)^{***}$
log(src_churn + 1)	$-0.06\ (0.01)^{***}$	$-0.06\ (0.01)^{***}$
log(files_changed + 1)	$0.01\ (0.02)$	$0.01\ (0.02)$
prs_main_team_member1	$0.06\ (0.07)$	$0.05\ (0.07)$
log(prs_popularity + 1)	$0.06\ (0.01)^{***}$	$0.07\ (0.01)^{***}$
log(num_issue_comments + 1)	$-0.25\ (0.01)^{***}$	$-0.24\ (0.01)^{***}$
prs_watched_repo1	$0.04\ (0.03)$	$0.05\ (0.03)$
prs_followed_pri1	$0.11\ (0.03)^{**}$	$0.10\ (0.03)^{**}$
prs_locationunited kingdom	$0.13\ (0.04)^{**}$	$0.20\ (0.04)^{***}$
prs_locationgermany	$-0.25\ (0.04)^{***}$	$-0.16\ (0.05)^{***}$
prs_locationfrance	$0.02\ (0.06)$	$0.11\ (0.06)$
prs_locationcanada	$0.12\ (0.07)$	$0.22\ (0.07)^{**}$
prs_locationjapan	$0.25\ (0.08)^{***}$	$0.34\ (0.08)^{***}$
prs_locationbrazil	$-0.27\ (0.06)^{***}$	$-0.19\ (0.07)^{**}$
prs_locationaustralia	$0.05\ (0.07)$	$0.14\ (0.07)$
prs_locationnetherlands	$0.26\ (0.09)^{**}$	$0.36\ (0.09)^{***}$
prs_locationchina	$-0.39\ (0.09)^{***}$	$-0.27\ (0.10)^{**}$
prs_locationrussia	$-0.06\ (0.07)$	$0.04\ (0.07)$
prs_locationspain	$0.08\ (0.10)$	$0.15\ (0.10)$
prs_locationindia	$0.02\ (0.07)$	$0.12\ (0.07)$
prs_locationswitzerland	$0.38\ (0.11)^{***}$	$0.46\ (0.11)^{***}$
prs_locationsweden	$-0.21\ (0.09)^{*}$	$-0.10\ (0.09)$
prs_locationbelgium	$0.09\ (0.12)$	$0.18\ (0.12)$
prs_locationitaly	$-0.31\ (0.08)^{***}$	$-0.21\ (0.09)^{*}$
prs_pri_same_location1		$0.18\ (0.03)^{***}$
AIC	49231.59	49198.20
BIC	49534.09	49509.87
Log Likelihood	−24582.80	−24565.10
Deviance	49165.59	49130.20
Num. obs.	70740	70740

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

the submitters and abiding by technical norms are seen to increase the chances of pull request acceptance while an increase in experience decreases it. We also notice that submitters with a high social reputation and those who follow social norms are more likely to get their pull requests accepted. Finally, the uncertainty associated with the pull request influences the chances of pull requests getting accepted. The influence of the above-mentioned factors is already known to the software engineering community [106][113]. Controlling for the effect of these factors in Model 1 in Table 8, we see that the geographical location of submitters explains significant differences in the chances of the pull request acceptance. The deviance explained by each factor in shown in Table 9.

Our observations support our hypothesis that there are differences in the pull request acceptance decision based on the geographical location of submitters (refer Table 8). Next, we identify the differences in the pull request acceptance decision location-wise. We see the sign and magnitude of the estimate for a given geographical location relative to the base level - the United States. The coefficients for submitters' geographical location can be divided into three categories: statistically insignificant coefficients, positive coefficients, and negative coefficients (refer Table 8). For 7 out of the 17 geographical locations under analysis, the results are statistically insignificant, that is, there are no differences in the chances of a contribution acceptance compared to the United States.

Table 9: Deviance explained by factors influencing pull request acceptance decision

	Df	Deviance	Resid. Df	Resid. Dev	Pr(>Chi)
NULL			70739	52354.02	
repo_pr_tenure_mnth	1	509.46	70738	51844.56	0.0000
repo_pr_popularity	1	175.49	70737	51669.07	0.0000
perc_external_contribs	1	321.47	70736	51347.59	0.0000
test_lines_per_lloc	1	47.84	70735	51299.75	0.0000
log(sloc + 1)	1	30.85	70734	51268.90	0.0000
prs_tenure	1	25.46	70733	51243.45	0.0000
log(prev_pullreqs + 1)	1	1014.64	70732	50228.81	0.0000
prs_succ_rate	1	389.62	70731	49839.19	0.0000
test_inclusion	1	22.73	70730	49816.46	0.0000
log(src_churn + 1)	1	164.02	70729	49652.44	0.0000
log(files_changed + 1)	1	0.11	70728	49652.34	0.7444
prs_main_team_member	1	2.62	70727	49649.72	0.1055
log(prs_popularity + 1)	1	57.46	70726	49592.26	0.0000
log(num_issue_comments + 1)	1	273.45	70725	49318.81	0.0000
prs_watched_repo	1	2.82	70724	49315.99	0.0931
prs_followed_pri	1	6.79	70723	49309.20	0.0092
prs_location	16	143.61	70707	49165.59	0.0000
prs_pri_same_location	1	35.39	70706	49130.20	0.0000

For geographical locations, where the coefficients are positive, the chances that their pull requests get accepted are higher than that of the United States. Similarly, geographical locations for which coefficients are negative have lower chances of getting their pull requests accepted. Thus, France, Australia, Russia, Spain, India, Sweden and Belgium observe no differences in getting their pull requests accepted compared to the United States. The United Kingdom (22%), Canada (25%), Japan (40%), Netherlands (43%), and Switzerland (58%) have higher chances of getting their pull requests accepted. Germany (15%), Brazil (17%), China (24%), and Italy (19%) have lower chances of getting their pull requests accepted. In Table 9, we see that the geographical location of submitters explains a small, yet significant percentage of the total deviance.

H2: There are differences in the pull request acceptance decisions based on the same geographical location of submitters and integrators.

To test this hypothesis, we control for the effect of above-mentioned factors including the geographical location of submitters. In Model 2 of Table 8, we see that the same geographical location of submitters and integrators, compared to different geographical locations of submitters and integrators, has statistically significant influence on pull request acceptance decisions. This observation supports our hypothesis 2. We see that controlling for the effects of other factors when submitters and integrators are from the same geographical location, there are 19% more chance that the pull requests will get accepted compared to when submitters and integrators are from the different geographical location. Further, in Table 9, we see that the same geographical location of submitters and integrators explains a small, yet significant percentage of the total deviance.

3.4.2 *Analysis of Survey Data*

3.4.2.1 *Submitters*

We received 1,603 complete responses from submitters. These responses present perspectives of a wide range of submitters from 76 countries with different age [21-30 years (47%), 31-40 years (42%), 41-50 years (7%) and others] , gender [male (98%), female (2%)], experience in OSS [1-2 years (10%), 3-6 years (53%), 7-10 years (19%), more than 10 years (17%) and others], job [Industry (62%), Academia (5%), Freelance (7%) and others], and role [source code contributor (49%), owner (47%) and others]. We analyse the perceptions of these submitters to answer our hypothesis H3.

H3: Submitters perceive bias based on their geographical location.

Table 10: Submitters' perception on bias - location-wise [AUC: 0.9]

	Bias
(Intercept)	4.84 (0.50)***
prs_locationGermany	0.01 (1.12)
prs_locationUnited Kingdom	16.73 (2908.74)
prs_locationFrance	16.73 (3116.19)
prs_locationCanada	−0.74 (1.13)
prs_locationRussia	16.73 (3906.35)
prs_locationBrazil	−2.60 (0.69)***
prs_locationAustralia	16.73 (4015.38)
prs_locationIndia	−1.20 (1.13)
prs_locationNetherlands	16.73 (4941.18)
prs_locationItaly	−2.13 (0.89)*
prs_locationSweden	16.73 (5250.30)
prs_locationSpain	16.73 (5524.41)
prs_locationJapan	−2.80 (0.79)***
prs_locationNorway	−1.75 (1.14)
prs_locationPoland	−1.75 (1.14)
prs_locationSwitzerland	16.73 (6379.04)
prs_locationUkraine	−2.07 (1.15)
prs_locationDenmark	16.73 (7812.70)
prs_locationCzech Republic	16.73 (8107.62)
prs_locationFinland	16.73 (8107.62)
prs_locationPortugal	−2.35 (1.16)*
prs_locationBelgium	16.73 (8438.68)
prs_locationArgentina	−2.54 (1.16)*
prs_locationAustria	16.73 (8813.91)
prs_locationIreland	16.73 (8813.91)
prs_locationSingapore	16.73 (9244.11)
AIC	236.25
BIC	378.45
Log Likelihood	−91.12
Deviance	182.25
Num. obs.	1432

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

There are statistically significant differences in the perceptions of submitters on the presence of bias. Using chi-square test of independence at a significance level of 0.05 (chi-squared = 1448.743, df = 1, p-value < 2.2e-16) we found that 97% more developers feel that they did not experience bias compared to those who feel that they experienced it. Our hypothesis H3, that is, submitters perceive bias based on their geographical location is rejected. Further analysis showed that the differences in the perceptions of bias across geographical location are statistically significant (chi-squared = 72.1569, df = 26, p-value = 3.203e-06), that is, there are differences in the perceptions of bias based on the geographical location of submitters. To tease out the individual effects, we modeled the perceptions on the experience of bias location-wise using logistic regression. An analysis of 27 geographical locations from which we received at least 10 survey responses shows that more submitters from some geographical locations perceive the presence of bias compared to others. We found that the perceptions of the presence of bias are more for submitters from Brazil (93%), Italy (87%), Japan (94%), Portugal (90%), and Argentina (93%) compared to other geographical locations (refer Table 10).

3.4.2.2 *Integrators*

We received 929 complete responses from integrators from 61 different geographical locations. These respondents were diverse in terms of age [21-30 years (38%), 31-40 years (46%), 41-50 years (12%)], gender [male (97%), female (3%)],
experience in OSS [1-2 years (6%), 3-6 years (43%), 7-10 years (24%), more than 10 years (26%) and others] , job [Industry (66%), Freelance (10%), Academia (6%) and others] and role [Owner (88%) and Source code (11%)]. We started the survey by understanding their perceptions of the level of awareness of the geographical locations of the submitters they work with. 43% of integrators say that

they are rarely aware of the geographical location of submitters they work with. This is followed by 28% of integrators who feel that they sometimes know the geographical location, followed by 16% who never know, 10% often and only 3% always. This was followed by a direct question to understand their perceptions on the importance of geographical location on the pull request acceptance decision for which we tested hypothesis H4.

H4: Integrators perceive that the geographical location of submitters is important on their pull request acceptance decisions.
88% more developers perceive that the geographical location of submitters is unimportant than those who consider it important (chi-squared = 705.8317, df = 1, p-value $<$ 2.2e-16) and this was consistent across all geographical locations (chi-squared = 12.958, df = 16, p-value = 0.6758). Thus, our observation refutes our hypothesis H4. This was followed by more specific questions to understand the influence of geographical location.

H5: Developers from some geographical locations are better at writing pull requests relative to others.
530 integrators responded to this question. An analysis of the integrators show with statistical significance that 52% more integrators disagree than those who agree that developers from some geographical locations are better at writing pull requests than others (chi-squared = 138.1231, df = 1, p-value $<$ 2.2e-16). This refutes our hypothesis H5 that developers from some geographical locations are better at writing pull requests relative to others. On digging deeper, we found that the perceptions are similar across geographical locations (chi-squared = 18.3874, df = 11, p-value = 0.07302), with an exception of India, where integrators felt that developers from some geographical locations are better at writing pull requests. For the rest of the geographical locations, for every 1 developer that feels that there are differences in the abilities of submitters to write pull requests based on geographical locations, there are 4 developers that disagree with it. On a contrary, half of the integrators from India agree and the other half disagrees with it (refer Table 11).

H6: Integrators perceive that it easy to work with submitters from the same geographical location.
418 integrators responded to this question. Integrators felt that it is easy to work with submitters from their own geographical location with statistically significant results (chi-squared = 80.597, df = 1, p-value $<$ 2.2e-16). There were 45% more integrators who felt that it is easy to work with submitters from their own geographical location. This supports our hypothesis H6. Further, we examined perceptions across geographical locations and found a different perspective (chi-squared = 24.4768, df = 8, p-value = 0.001906). On one side, countries like the United States (8 out of 10) and the United Kingdom (6 out of 10) agree that it is easy to work with developers from the same geographical location. On the other side, integrators from Germany and India disagree 6 out of 10 and 8 out of 10 times respectively. In short, a majority of the geographical locations analysed feel that it is easy to work with developers from the same geographical location with few exceptions (refer Table 12).

H7: Integrators encourage submitters from their geographical location to participate.
437 integrators responded to this question. 26% more integrators agree that they encourage developers from their geographical location to participate with statistically significant results (chi-squared = 27.8244, df = 1, p-value = 1.328e-07). This supports our hypothesis H7. Further, we see that perceptions on this depend on the geographical location of integrator (chi-squared = 24.3008, df = 8, p-value = 0.00204). Unlike other geographical locations, integrators in the United Kingdom (62%), Germany (69%) and Sweden (77%) are less likely to encourage developers from their geographical location to participate (refer Table 13).

3.4.3 *Open-ended Survey Responses*

We received 639 open-ended survey responses from integrators where they talked about factors that influence their acceptance decisions. From manually coding these open-ended responses, six themes came up: *technical skills (47%), the relevance of the requested feature (12%), communication skills (23%), behaviour (11%), trust (4%), and pro-activeness (2%).* We went deeper into non-technical aspects to uncover the expectations of integrators that may possibly explain the perceived differences in pull request acceptance decision. One key observation from these responses was the ability to communicate. Integrators mentioned that they form an impression about the pull request based on the description

Table 11: Integrators' perception on pull request quality and geographical location [AUC: 0.6]

	Better loc on PR
(Intercept)	$1.30\ (0.16)^{***}$
pri_locationUnited Kingdom	$0.84\ (0.55)$
pri_locationGermany	$0.22\ (0.45)$
pri_locationFrance	$-0.52\ (0.52)$
pri_locationCanada	$-0.04\ (0.59)$
pri_locationSweden	$1.19\ (1.05)$
pri_locationAustralia	$-0.71\ (0.58)$
pri_locationNetherlands	$-0.09\ (0.68)$
pri_locationIndia	$-1.45\ (0.58)^{*}$
pri_locationBrazil	$0.90\ (1.07)$
pri_locationSwitzerland	$1.27\ (1.05)$
pri_locationNorway	$-0.45\ (0.71)$
AIC	440.03
BIC	488.51
Log Likelihood	-208.01
Deviance	416.03
Num. obs.	420

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

Table 12: Integrators' perception on ease to work with the same location developers [AUC: 0.7]

	In group ease
(Intercept)	$1.50\ (0.19)^{***}$
pri_locationUnited Kingdom	$-1.10\ (0.45)^{*}$
pri_locationGermany	$-1.76\ (0.46)^{***}$
pri_locationFrance	$-0.55\ (0.56)$
pri_locationCanada	$1.14\ (1.05)$
pri_locationSweden	$15.07\ (665.51)$
pri_locationAustralia	$0.11\ (0.80)$
pri_locationIndia	$-2.89\ (0.81)^{***}$
pri_locationBrazil	$-1.50\ (0.84)$
AIC	323.24
BIC	356.99
Log Likelihood	-152.62
Deviance	305.24
Num. obs.	314

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

of the pull request. Integrator [R496] mentioned that *"bad or misleading title influence her chances of accepting the pull request"*.

They also used the ability to communicate to justify their perceptions of the importance of the geographical location of submitters in explaining pull request acceptance decision. Integrator [R289] stated that *"Nationality really only plays a role in my ability to understand the written communication. Being distributed makes communication more important, so if I cannot understand the requester<submitter>, I'm much less likely to accept the request."*. Further, integrator [R201] added that *"There's a possibility of language bias - if the pull request isn't well-written (which is often the case when English is not the PR author's first language) I may be more hesitant, but usually because of a fear of misunderstanding."*. Integrator [R883] even goes on to add that *"I often reject pull requests that add in code that has misspelled words, poor grammar, etc., and ask the contributor to fix those before merging."*. In addition to this, integrators also mentioned the importance of behaviour, trust, and pro-activeness of submitters on their pull request acceptance decision. Integrator [R738] said that the tone of the pull request's body is important for her. She said that *"I don't want to work together if the person is rude."*. Integrator [R773] even goes on to say that *"If they<submitters> are rude, their pull request is rejected, even if the code quality is great."*. Besides these, integrators place their bet on the submitters they trust. Integrator [R50] stated that *"People that have submitted good PRs in the past I almost blindly merge"*. Finally, integrators appreciate the willingness of the submitters to quickly make desired changes to improve contribution. Integrators ruled out the

Table 13: Integrators' perception on encouraging developers from same geographical location [AUC: 0.6]

	In group encourage
(Intercept)	$0.81\ (0.16)^{***}$
pri_locationUnited Kingdom	$-1.00\ (0.40)^{*}$
pri_locationGermany	$-1.18\ (0.42)^{**}$
pri_locationFrance	$-0.19\ (0.50)$
pri_locationCanada	$0.20\ (0.61)$
pri_locationSweden	$-1.50\ (0.63)^{*}$
pri_locationNetherlands	$-0.25\ (0.65)$
pri_locationIndia	$0.17\ (0.70)$
pri_locationBrazil	$15.76\ (758.80)$
AIC	399.35
BIC	433.04
Log Likelihood	-190.67
Deviance	381.35
Num. obs.	312

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

presence of explicit bias based on the geographical location in favour of the valuable contribution they receive for their projects. In this context, integrator [R661] quoted that *"I see PRs as a favor to me, so I tend to take it seriously to process PRs ASAP and treat requesters<submitters> well."*.

3.5 THREATS TO VALIDITY

3.5.1 *Internal Validity*

Data accuracy The accuracy of the results of the study depends on the accuracy of the data on which it is built. We have used GHTorrent data, which has been extensively used in several prior studies.

Language bias The surveys deployed for this study were written in English. We justify our choice by conducting a pilot study where an equal number of surveys were sent out in English and French to developers in France. We received similar response rates from both. This is intuitive as developers who use GitHub must be aware of basic English used in the survey. Still, there is a possibility that the choice of English biased the response rates from some geographical locations.

Researcher bias To prevent researcher's bias on the articulation of our questions, we got our survey questions validated by a wide range of people including survey design experts even before making it public. These people checked the language of the questions for ambiguity and the presence of bias.

Research reactivity The tendency of the respondents to appear in the positive light may influence the results. However, we tried to minimize it by conducting anonymous surveys and not giving any monetary incentive to survey respondents.

Non-response bias It is possible that the developers who did not respond to the survey may have different insights. However, we do not see this as a big concern as we received survey responses from more than 76 geographical locations.

3.5.2 *External Validity*

Generalisability The quantitative analysis present in this study is built on small, carefully selected projects. These projects may not be representative of all the collaborative developed projects. We try to address this concern by combining the results of the quantitative data with large-scale survey responses. Further, while we conducted this survey on GitHub - one of the biggest code hosting sites featuring pull-based development model, we believe that similar experiments must be conducted on other platforms, like Bitbucket, for generalisability.

3.6 IMPLICATIONS

Data analyses suggest the presence of geographical bias. Analyses of 70,000+ pull requests show that the geographical location of submitters significantly influences the pull request acceptance decisions. Compared to the United States, submitters from United Kingdom (22%), Canada (25%), Japan (40%), Netherlands (43%), and Switzerland (58%) have higher chances of getting their pull requests accepted. However, submitters from Germany (15%), Brazil (17%), China (24%), and Italy (19%) have lower chances of getting their pull requests accepted. Also, the same geographical location of submitters and integrators increases the chances of pull request acceptance by 19%.

Submitters from some geographical locations perceive to experience bias. Overall, submitters do not perceive that they experience bias. 97% more submitters feel that they did not experience bias compared to those who felt that they experienced it. However, submitters from some geographical locations perceive to have experienced bias, which is more compared to the other geographical locations. We found that the perceptions of the presence of bias are stronger for submitters from Brazil (93%), Italy (87%), Japan (94%), Portugal (90%), and Argentina (93%) when compared to other geographical locations.

Perceptions of submitters on geographical bias are in agreement with the data analysis. Submitters from Brazil and Italy perceive to experience bias more than other geographical locations. The same is observed in our analysis of GitHub data that developers from Brazil and Italy have lower chances of getting their pull requests accepted compared to other geographical locations.

Integrators perceive that they are not biased in evaluating submitters. 53% more integrators perceive that they encourage developers from their country to participate. Also, 8 out of every 10 integrators feel that it is easy to work with submitters from the same geographical location. However, they do not feel that developers from some geographical locations are better at writing pull requests compared to others, with an exception of India. For every 1 developer who feels that there are differences in the abilities of submitters to write pull requests based on geographical location, there are 4 developers that disagree with it. However, in India half of the integrators agree and the other half of the integrators disagree with it.

Integrators think that factors relating to the geographical location and not necessarily the geographical location may influence their pull request acceptance decisions. In the open-ended survey, integrators suggest that the observed differences may be explained in terms of language barriers and the ability to communicate, and not necessarily bias based on the geographical location of submitters.

There exists a bias blind spot [136] - a cognitive bias, as integrators perceive the absence of bias and submitters perceive to experience it. Submitters from some geographical locations perceive to experience bias and is supported by data analysis. This is contrary to the opinions of integrators that they are biased. This may imply the presence of geographical bias with disagreement in the perceptions of integrators. Alternatively, it may be due to communication barriers which make submitters experience bias while integrators are just trying to ensure quality submissions.

We have various reasons why we believe that it is not due to communication barriers: 1) There are geographical locations in the list which have higher chances of pull requests acceptance and where English is not widespread in use (relatively speaking) like Japan. 2) Switzerland and Germany are pretty much exactly similar in terms of language use, but fall in different spectrums. While English is widely spoken and used in Germany it has lower chances of pull request acceptance. 3) We sent the survey requests in different languages like French and got numerous responses that stated that all GitHub developers communicate through English and we should not translate survey and send in the local languages. 4) The law of large numbers also ensures that our results are not biased by a handful of people who have poor communication skills. Together these suggest the actual presence of geographical bias and a bias blind spot as developers see the impact of bias on others judgement while failing to see the impact of bias on their own judgement.

3.7 SUMMARY

In this chapter, we combined observations from 70,000+ pull requests and 2,500+ survey responses to analyze the influence of geographical location on pull request acceptance decisions. We found a bias blind spot- a type of cognitive bias in the peer code review process in GitHub. This study shows that integrators perceive the absence of bias while submitters from some geographical locations perceive to experience it. The perceptions of submitters match the data analysis - differences in the pull request acceptance rates based on geographical locations. This study informs integrators and submitters about the presence of bias despite the differences in their reported perceptions.

4 FACTORS INFLUENCING CONTRIBUTOR PARTICIPATION

Managers and decision makers are interested in the explanations of contributor productivity. These explanations facilitate planning and informed decisions. In this chapter, we study the influence of various project and contributor characteristics on contributor participation. We start with a study to understand how competing projects' dynamics influence contributor participation. Next, we study the relationship between the personality traits of contributors and their levels of contribution. We also study that how people perform in different work situations. Finally, we see that how role and past contributions relate to future participation. A detailed description of each study is presented in the following sections.

4.1 DYNAMICS OF COMPETING PROJECTS

4.1.1 *Introduction*

Sustained developer community participation is fundamental to the long-term success of open source software (OSS) projects [137]. A significant fraction of OSS projects fails due to their inability to attract a critical mass of developers [138] and lack of sustained developer community participation [63]. The problem proliferates when projects start competing for the developer community participation [139].

In OSS, new projects may start using someone else's project as a starting point, a phenomenon termed as *forking* of the original project. The new project emerged, termed as *independently developed fork (IDF)*, competes with the original project for the developer community participation. The competition is not only for the existing developer community participation, but also for the potential developer community participation to ensure sustained development of the project.

Some recent examples of popular forks are 1) the start of LibreSSL out of frustration with OpenSSL[1], 2) fork of IO.js from Node.js, which recently got merged into the original[2], and 3) Docker fork created by Windows[3]. These popular, competing projects gained the attention of the masses, making it relevant to understand how the original and the forked projects are doing in terms of developer involvement. Besides these, many not-so-popular and small projects observe forking. Thus, to better understand the project's development trajectory and success, there is a need to understand the response of the developer community to this change. An understanding of the response of the developer community to forking may inform future research to support preemptive measures for project success.

Forking has received mixed reactions from the developer community [140]. Existing studies claim that forking is healthy for the software ecosystem in the 'survival of the fittest sense' [141][142][143]. However, the split (in the project) costs the loss of developers as the synergy to work together is lost [141][142][143] and incompatibilities among the two projects [144]. Other studies visually analyzed temporal trends in large, popular projects, like OpenOffice and its fork LibreOffice and observed that forking has no effect on the original project and the emerged IDF [137][139]. However, a comprehensive understanding of the influence of forking is missing. In this study, we statistically investigate the influence of forking on the sustainability of the developer community participation in a wide range of projects. The relevance of asking this research question (RQ) increases with the recent changes in the software development, that is, the use of coding sites like GitHub, Bitbucket, etc.

GitHub is a project-hosting site build on distributed version control system. Currently, it is the most popular coding site, surpassing GoogleCode and SourceForge in popularity[4]. GitHub hosts more than 23.8 million git repositories[5] (as observed in June 2015) and has more than 6 years of development history. With its 1) widespread popularity, 2) representation of a wide range of projects in terms of size, language, domain, etc., and 3) availability of the data through GitHub REST API for analysis,

1 http://arstechnica.com/information-technology/2014/04/openssl-code-beyond-repair-claims-creator-of-libressl-fork/
2 http://news.softpedia.com/news/io-js-Will-Be-Merging-Back-into-Node-js-482552.shtml
3 http://blog.docker.com/2014/10/docker-microsoft-partner-distributed-applications/
4 http://readwrite.com/2011/06/02/github-has-passed-sourceforge
5 https://github.com/features

GitHub has emerged as a favored test-bed for conducting a wide variety of large scale studies [145] [146]. In this study, we leverage the diverse projects' data hosted on GitHub to answer the following:

How forking influences the sustainability of the developer community participation in the original project hosted on GitHub?

In GitHub, many active projects are brought from outside, dating earliest to 1999. These *imported projects* are mirrors of projects hosted elsewhere or are ported from git or other version control systems like Mercurial after some initial development. Unlike the projects that are created on GitHub, also termed as *internally developed projects*, these projects have a prior development history and assumably have reasonably mature code at the time of project creation on GitHub. In this study, we study the two classes of projects in parallel and look for possible differences in their response to forking. So, the sub research question we try to address is:

Do projects ported to GitHub behave differently relative to projects created in GitHub?

Understanding the community response to forking alone does not help project success. There is a need to understand the factors driving community behavior which can be fine-tuned to improve the chances of a project's success. So, here we examine the role of project characteristics in competing projects at the time of forking on the observed behavior. We ask the following question:

How do project characteristics at the time of forking influence the *forkability* and the sustainability of the developer community in the original project?

To investigate the influence of forking, we propose a quantitative framework to identify forks, characterize projects, and measure changes in the developer community participation in the original project after forking. To do so, we identify types of forks and propose a quantitative approach to single out independently developed forks from the original project – a concept measured in qualitative terms in literature. The proposed measure helps to analyze a wide range of projects to get a comprehensive view of the influence of forking. Next, we distinguish between internally developed and imported projects on GitHub using quantitative measures. This classification of projects may help future research on project characterization to examine the two distinct classes of projects distinctly. Thus, while designing the framework for experimentation, we also make the following two contributions in this study:

1. Classification of forks and identification of IDFs in quantitative terms.
2. Classify projects as imported and internally developed projects quantitatively.

We start with an early remark on the relevance of forking in GitHub by analyzing temporal trends. As we discuss later, the term *forking* is used in a broad manner in GitHub. So, unless stated otherwise, we use the term forking to imply the emergence of IDFs from the original project.

To measure the sustainability of the developer community participation, we adopt and extend approaches from existing studies and measure the contribution of the developer community in terms of software configuration management artifacts. We measure changes in the developer community participation after forking and model the influence as 1) decrease, 2) increase, and 3) no effect on the sustainability of the developer community participation in the original project. Further, we model the observed behavior in terms of project characteristics to understand its reasons.

A large-scale study of projects hosted on GitHub shows that forking significantly alters the sustainability of the developer community participation in the original project. We find that 18%, 29%, and 53% of the original projects observe *decrease*, *increase*, and *no effect* respectively on the developer community participation after forking. These results indicate that a small, yet significant, fraction of projects observe a decline in the sustainability of the developer community participation after forking. This effect is more pronounced in projects ported to GitHub from external sources than projects created in GitHub. We find that the maturity of the project, popularity of the owners of the competing projects, domain, community size of the competing projects, and popularity of the competing projects explain the forkability and the chances of the decline in the developer community participation.

4.1.2 *Related Work*

Forks may result from a wide variety of reasons. Web-searches of popular forks [147], interviews, surveys [148], and fact-sheets [149] created by researches found that forks may result from 1) technical reasons, 2) disregard for the developer community, 3) discontinuation of the original project, 4) commercialization of fork, 5) legal issues, and 6) differences among the team. Forking is observed in every domain and very few forks are merged into the original [147]. Literature on popular forks suggests that forking has no effect on the original project or the emerged new project[148] [149]. The above-mentioned studies examine the reasons and outcomes of forking through qualitative measures. However, the influence of forking on the developer community participation has seen little study. A closely related work by Gamalielsson *et al.* analyzed developer communities and their evolution in terms of project activity, developer commitment, and retention of committers over time [137]. They supported their analysis with the first-hand experiences of contributors [139]. This study visually analyzes temporal trends in a large, popular project and presents their observations. However, a comprehensive analysis of the influence of forking on a wide variety of projects is missing. Further, what explains the observed behavior is not known. There are studies that examine the role of individual factors (willingness, opportunity, identity construction, etc.), and social factors (sociality, situated learning, etc.) on long-term participation of developers [150] [59] [11]. However, the characterization of the observed behavior in terms of project characteristics is missing to the best of our knowledge.

4.1.3 *Background*

4.1.3.1 *Forks*

In terms of Eric Raymond, forking “spawns competing projects that cannot later exchange code, splitting the potential developer community" [151]. The definition of forking has evolved since then. In recent times, GitHub started using the term *forking* in a broad fashion. In GitHub, a repository is forked to 1) propose changes to someone else’s project and 2) use someone else’s project as a starting point of a new idea[6]. Based on the current usage of the term ‘fork’ in GitHub, there are three types of forks:

1. *Contributing forks:* Forks which propose changes to others’ code.
2. *Independently developed forks:* Forks which use someone else’s project as a starting point to a new idea.
3. *Inactive forks:* Forks which neither contribute nor are maintained independently.

All forks in GitHub can be classified into one of the three mutually exclusive categories based on their contributions. In GitHub, there are two ways to collaborate.

1. In the *Fork & Pull model* changes are reviewed via pull requests before they are included in the main project.
2. In the *shared repository model*, everyone is granted push access to a single shared repository and topic branches are used to isolate changes.

For this study, we choose the Fork & Pull model – a popular collaborative development model in open source projects. The choice is also driven by the fact that the shared repository model is prevalent in 1) private projects which are not analyzed in this study, and 2) small teams which are excluded from analysis, as we discuss later. Thus, we limit the scope of this study to Fork & Pull collaborative development model. Now, we introduce the basic terminology used for forks, followed by defining the rules for classification. Finally, we classify forks to identify original projects, independently developed

6 https://help.github.com/articles/fork-a-repo/

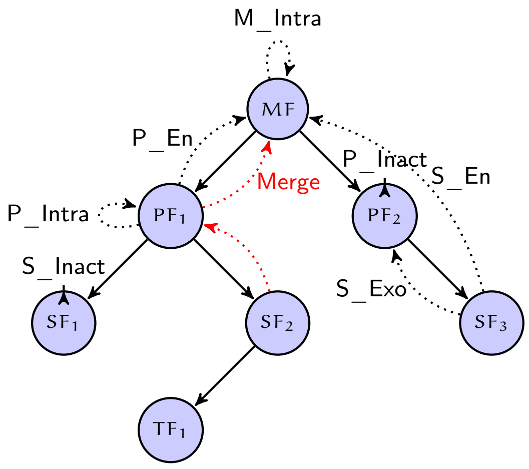


Figure 14: Fork Tree, and Types of Forks on Pull Request History

forks, and inactive forks. The proposed framework can be used to identify forking in projects of different sizes quantitatively.

BASIC TERMINOLOGY Forking in GitHub follows a tree structure [152] with the original project, also termed as the *master fork (MF)*, at the root of the tree. All subsequent forks of the master fork are termed as *primary forks (PFs)*, *secondary forks (SFs)*, *tertiary forks (TFs)* and so on [152]. In Fig. 14, the original project, MF is at level 0. PF_1 and PF_2 are forked from MF and are at level 1. Similarly, SFs forked from PFs and TFs forked from SFs are at level 2 and level 3 respectively. Unlike forks which follow a tree structure, pull requests and commits float arbitrarily by appropriately setting remotes[7] to relevant repositories. This arbitrary contribution behavior decides the relationship between forks and hence the rules for classification discussed below.

RULES FOR CLASSIFICATION For projects build using *Fork & Pull model*, following are the rules for classification:

- Forks that initiate pull requests to predecessor forks are *contributing forks*.
- Forks that do not send pull requests to predecessor forks, but observe internal commits are *independently developed forks*. These forks may get pull requests from successor forks.
- Forks that neither send nor receive pull requests nor have internal commits are *inactive forks*.

Based on the rules defined above, the original project is an independently developed fork with successor IDFs. The proposed definition of original projects and independently developed forks is inclusive of various possible combinations. Thus, in a sub-system PF can act as the original project for SF, if 1) PF does not contribute back to the MF and is developed independently with pull requests from successor forks and internal commits, and 2) SF (derived from PF) does not contribute back to the PF or MF and is developed independently. However, for the ease of analysis and considering the limited data points at higher levels in the tree, we limit the definition of the original project to MF and that of the independently developed fork to PF.

NOMENCLATURE AND CLASSIFICATION With MF as the original project and PF as the only possible independently developed fork, we present a detailed nomenclature for forks. This is followed by the classification of forks based on all possible contribution patterns:

1. Forks which send pull requests to IDFs (PFs) are termed as *exogenous forks* while forks which send pull requests to original projects (MFs) are termed as *endogenous forks* [152].

7 https://help.github.com/articles/using-pull-requests/

Table 14: Classification of Forks [*PR*: Pull Request; *MF*: Master Fork; *PF*: Primary Fork; *SF*: Secondary Fork]

Category	Symbol	Name	Description	Pull Request	Commit History
Independently Developed Forks	S_**Exo**	Secondary Exogenous Fork	**PF** $\xleftarrow{PR}$ SF	✓	Not Required
	P_**Intra**	PF Intra	**PF** $\xleftarrow{PR}$ PF	✓	Not Required
	P_ID	Internally Developed	**PF** with commit history	✗	✓
Contributing to the Master Fork	P_**En**	Primary Endogenous Fork	**MF** $\xleftarrow{PR}$ PF	✓	Not Required
	S_**En**	Secondary Endogenous Fork	**MF** $\xleftarrow{PR}$ SF	✓	Not Required
	Merge	Merged Fork	**MF** $\xleftarrow{PR}$ PF $\xleftarrow{PR}$ SF	✓	Not Required
	M_**Intra**	MF Intra	**MF** $\xleftarrow{PR}$ MF	✓	Not Required
Inactive Forks	**P**_Inact	Primary Inactive Fork	**PF** with no pull request(s) or commit history	✗	✗
	S_Inact	Secondary Inactive Fork	**SF** with no pull request(s) or commit history	✗	✗

a) Secondary forks (SF) which send pull requests (PR) to Primary Forks (PF) are termed as *secondary exogenous forks* (see S_Exo = $PF_2 \xleftarrow{PR} SF_3$ in Fig. 14 and Table 14).

b) Pull requests by PFs and SFs to MFs are termed as *primary endogenous forks* (P_En) and *secondary endogenous forks* (S_En) respectively (see P_En = MF $\xleftarrow{PR}$ PF_1 and S_En = MF $\xleftarrow{PR}$ SF_3 in Fig. 14 and Table 14).

2. There is one more variant to this category where SFs send pull requests to PFs and PFs send pull requests to MFs. These forks are termed as *merged forks* (see Merge = MF $\xleftarrow{PR}$ $PF_1 \xleftarrow{PR} SF_2$ in Fig. 14 and Table 14).

3. When the source and destination of pull requests are same, it is *intra branch* pull requests. Pull requests are termed as *MF Intra* (M_Intra) and *PF Intra* (P_Intra) when the source and destination are MFs and PFs respectively (see M_Intra = MF $\xleftarrow{PR}$ MF and P_Intra = $PF_1 \xleftarrow{PR} PF_1$ in Fig. 14 and Table 14).

4. Forks which do not send pull requests (refer column Pull Request in Table 14) are classified into two heads on commit history (refer columns Pull Request and Commit History in Table 14).

   a) Forks with no pull requests, but commit history are *internally developed forks* (see P_ID in Table 14).

   b) Forks with no pull requests and no commit history are *inactive forks* (see P_Inact and S_Inact in Table 14).

The definition descends for all original project-independently developed forks pair. So, if PF acts as the original project, pull requests from SFs to PFs are SF Endogenous Fork and pull requests from TFs to independently developed SFs are Tertiary Exogenous Fork. For this study, secondary exogenous fork (S_Exo), primary fork intra (P_Intra) and internally developed (P_ID) are *Independently Developed Forks* (refer Section 1 in Table 14). Similarly, primary endogenous fork (P_En), secondary endogenous fork (S_En), merged fork (Merge) and MF Intra (M_Intra) are contributing forks *contributing to the MF* (refer Section 2 in Table 14). Also, primary inactive fork (P_Inact) and secondary inactive fork (S_Inact) are *Inactive Forks* (refer Section 3 in Table 14).

4.1.3.2 *Projects Ported vs Created on GitHub*

In GitHub, a wide variety of projects are ported from elsewhere. Recently, GoogleCode announced its shutdown and ported numerous projects to GitHub[8]. In this section, we present a quantitative approach to distinguish between projects created in GitHub and projects ported from elsewhere.

8 http://google-opensource.blogspot.in/2015/03/farewell-to-google-code.html

Imported projects include mirrors of popular projects which observe development activities at their respective official sites [153]. These projects with no development within GitHub are not of any interest for this study. And interestingly, these projects characterized by no forks or fork tree structure [153] are automatically eliminated when we select projects with fork tree structure at least up to two levels. The next category is of projects ported from git or other version control systems. These projects have a fair amount of code before they are ported to GitHub.

One characteristic that differentiates imported projects from internally developed projects is their commit history. Imported projects have commit history prior to their date of creation on GitHub. Manual analysis of projects hosted on GitHub shows development history dated back to the year 1999. We use this feature to separate the two classes of projects.

1. *Imported projects:* Projects with commit history prior to project creation date on GitHub.
2. *Internally developed projects:* Projects with commit history (or pull request history) after project creation date on GitHub.

It is important to note here that imported projects do not have pull request history, so we compare commit history with project creation date on GitHub.

4.1.4 *Methodology*

We identify projects with independently developed forks and statistically analyze changes in the developer community participation in the original project after forking. A step-by-step detailed description of the methodology is described hereby. To help replication, the data are made publicly available[9].

4.1.4.1 *Data Acquisition and Curation*

We start with the GitHub mysql data[10] made available at GHTorrent on January 2014 [154]. GHTorrent is an offline mirror of the data offered through the GitHub REST API. It monitors the GitHub public event timeline and tracks all user activities and events in a project, including commits, pull requests, etc. The downloaded GHTorrent data stores development history of 3,004,002 projects hosted on GitHub from October 2007 to January 2014.

PREPROCESSING

- *Project Selection Criteria:* We select projects where MFs are created between the year 2008 and 2012. The choice of 2012 as the end year ensures that we have at least a year of development history, even for the projects that are created towards the end of the year 2012.
- *Project Elimination Criteria:* We eliminate projects with deleted MFs. Public projects where the MF is deleted, are assigned a new MF and all subsequent forks and pull requests go to the new MF[11]. The distribution of the developer community contribution into old and new MF gives ground to remove such projects from the study.

IDENTIFY PROJECTS WITH IDFS Selection of projects with IDFs require that we select projects with at least two levels in the fork tree. This step also eliminates mirrored projects that are actively maintained outside GitHub and have no forks in GitHub (Peril IX) [153]. Thus, we select projects with MF at level 0, PF at level 1 and SF at level 2. A two level fork tree captures all possible combinations defined in the scope of this study. Empirical analysis shows that this also ensures that we eliminate personal projects and select collaboratively developed MF in the study. Then, based on the fork classification criteria discussed in Section 4.1.3.1, we identify projects with independently developed forks.

We select projects where IDFs received significant participation from the community to possibly influence the original project. So, we select projects where IDFs have at least 100 commits, 90 days of activity, 4 contributors and 2 committers (an indicator of significant contribution used in [146]). This gives us MF-PF pairs where PFs have significant developer community participation to influence the original project. It is important to note here that we do not apply the significance criteria on the

9 https://github.com/AyushiRastogi/fsoc
10 http://ghtorrent.org/downloads.html
11 https://help.github.com/articles/what-happens-to-forks-when-a-repository-is-deleted-or-changes-visibility/

original project. This ensures that we do not bias our data set towards the original projects that get significant attention from the developer community. Now we have MF-PF pairs where one MF can have multiple PFs based on significant participation from the developer community. Here, we treat each MF-PF pair uniquely because different PFs may cast different influence on their MFs. Finally, we aggregate the results and present the cumulative effect for projects.

CLASSIFY PROJECTS ON DEVELOPMENT HISTORY BEFORE CREATION DATE ON GITHUB We classify the selected MF-PF pairs on prior development history. MF-PF pairs, where any of the MF or PF has commit history prior to their creation date on GitHub, are classified as imported projects. Rest all MF-PF pairs are classified as internally developed projects. We study the two classes of MF-PF pairs in parallel.

4.1.4.2 *Developer Community Participation*

All relevant contributions to a repository are bookmarked in the form of commits. GitHub's pull based development model facilitates the core contributors to push their changes directly into the repository while the rest submit their contributions via pull requests. The pull requests are reviewed and contingent on their suitability these requests may get merged into the system. Contributors can also report issues, post comments to help get it fixed, and close issues by committing[12]. Thus, commit counts provide a fair representation of the overall activities of contributors and has been used in various studies[137][146]. So, in this study, we measure commit counts as a proxy of developer community participation and use commit counts per time-interval to measure the sustainability of the developer community participation. It is important to note here that we do not measure changes in developer community rather changes in developer community participation. The justification for our choice is that developers may participate in both the competing projects. Also, developers may adjust their efforts to account for the developers' loss to ensure sustainability of the project.

4.1.4.3 *Influence of Forking*

For MFs of the selected projects, we identify breakpoint on the creation date of fork and study changes in the developer community participation after forking. One of the key challenges to empirically investigate projects with forks is sporadic developer community participation in the original projects. We found that the count of time-intervals when developer community participates is significantly less than the count of time-intervals of project existence. So, to ensure that we have 1) significant development history and 2) fair count of projects to conduct analysis, we experiment with multiple *time-intervals of existence* and *time intervals of activity*. In this study, we experiment with 3, 6, 9, and 12 month time intervals of existence and activities and present our results.

We model changes in the developer community participation using Mann-Whitney U Test, a non-parametric test and decide at the 0.05 significance level. The choice of non-parametric test is guided by the significant fraction of projects with insufficient developer community participation to approximate a normal distribution. Also, this ensures that contribution spikes during the time of release do not bias our results. The two-sided hypothesis to study changes in the developer community participation is as follows:

H_0 : Forking has *no effect* on the sustainability of the developer community participation in the original project.
H_a : Forking *changes* developer community participation in the original project.

Further, we measure the direction of change (increase or decrease) when forking changes developer community participation in the original project.

4.1.4.4 *Project Characteristics and Developer Community Participation*

Projects are characterized by various static (owner, domain, tools and technology used, etc.) and temporal (maturity, contributors, process followed, tools and technology used, popularity, etc.) attributes. In this study, we discuss the role of some of these characteristics in explaining the forkability and

12 https://help.github.com/articles/closing-issues-via-commit-messages/

sustainability of the developer community participation in the original project. Below, we define these project characteristics in terms of SCM artifacts.

MATURITY Maturity of the MF at the time of creation of PF is measured as the duration between the minimums of MF creation or first commit date and PF creation or first commit date. The choice of first commit date accounts for projects with development activities outside the GitHub development environment. Mathematically, age of MF is $\{\min(PF_{created}, PF_{firstCommit}) - \min(MF_{created}, MF_{firstCommit})\}$.

DEVELOPER COMMUNITY SIZE Developer community size is the count of contributors whose code was committed into the repository.

OWNER We measure the influence of the owner in terms of the nature and size of the follower base. By the term nature of followers, we mean the influence of the followers of the owner, measured here as the count of the follower's followers. Thus, the influence of the owner is $\sum follower[follower[repository]]$.

DOMAIN The domain of projects hosted on GitHub is not stated explicitly. In this study, we infer the domain of a project from its textual description. We use a mix of automated and manual techniques (as suggested by [4]) and classify projects into six domains, namely, application, database, compiler, middleware, library, framework, and others. We identify topics using Latent Dirichlet Allocation (LDA), a well-known topic analysis algorithm and then manually classify the topics into the six domains described above.

POPULARITY We measure the popularity of a repository in terms of watchers' count for the repository.

4.1.4.5 *Statistical Method*

We use logistic regression to model the relationship of a set of predictors (project characteristics) against a binary outcome, which is, significant fork or insignificant fork, and a decline in developer community participation or not. We choose each MF-PF pair as a row in our regression model. Our choice to consider each MF-PF pair as an independent observation is motivated by the fact that the dynamics of each PF with the same MF is different.

Descriptive statistics of the independent variables show that our data is *heteroscedastic*. Thus, to stabilize the variance and help improve the model fit, we log transform the data. We verify this by comparing the AIC of the model with transformed and non-transformed data. Further, to help compare 'domain' relative to base level, we use treatment code. As we expect the developer community participation to respond differently to different sized projects, we classify projects on team size as small, medium, and large-sized projects. Small sized projects have less than 11 contributors in MF, medium sized projects have more than 10 and less than 30 contributors, and large sized projects have more than 30 contributors as suggested by [146]. We build the regression model using generalized linear models, as implemented in the function *glm* from the package *glm2* in R. Coefficients are considered important if they are statistically significant ($p < 0.05$). Effect sizes are obtained from ANOVA analyses and we evaluate models using residual deviance as a measure of goodness of fit. Table 17 and Table 18 show the results from the study. Here the coefficients are read as the expected changes in the log of responses for a unit change in the predictor variable while keeping all other predictor variables at a constant value. Log odds can also be expressed as (exp(coeffs)-1)*100 percentage change in odds ratio. The interpretation of the result is the percentage change in odds (relative to the baseline) by which a unit increase in the predictor variable influence the response variable.

4.1.5 *Results*

Table 15 shows the count of projects selected in each step of the methodology discussed in Section 4.1.4. The projects count is measured annually to help analyze temporal trends. A total of 1,082,501 projects are hosted on GitHub from the year 2008 to the year 2012 where the projects count triples every year (refer 'Projects Count' in Table 15). Based on the project selection/elimination criteria in preprocessing,

Table 15: Basic statistics of GitHub dataset

Year	Projects Count	Selection/ Elimination Criteria	Collaboratively Developed Projects	Projects with Independently Developed Forks (MF-PF pairs)	Total Projects = Projects developed within ∪ outside GitHub with Significant Forks (MF-PF pairs)
2008	008,873	008,541	004,635	01,211 (028,216)	0,277 = 038 ∪ 0,263 (0,056 / 0,569)
2009	026,775	025,632	015,161	03,633 (067,087)	0,647 = 103 ∪ 0,615 (0,339 / 1,143)
2010	064,886	061,407	034,300	08,559 (123,471)	1,277 = 215 ∪ 1,196 (0,390 / 2,221)
2011	179,654	165,718	082,576	19,497 (199,127)	1,815 = 317 ∪ 1,670 (0,565 / 3,134)
2012	802,313	616,390	178,661	32,738 (219,719)	1,408 = 262 ∪ 1,223 (0,398 / 1,993)
Total	1,082,501	877,688	315,333	**65,638** (637,620)	**5,424** = 935 ∪ 4,967 (1,748 / 9,060)

we identified 877,688 projects out of which 315,333 projects have fork structure up to two levels and 65,638 projects with IDFs or 637,620 MF-PF pairs (refer Total of 'Selection/Elimination Criteria', 'Collaboratively Developed Projects' and 'Projects with Independently Developed Forks' respectively in Table 15).

Approx. one-third of the projects hosted on GitHub are developed collaboratively (refer columns 'Projects Count' and 'Collaboratively Developed Projects' in Table 15). This observation is in agreement with the study of Kalliamvakou *et al.* [153]. Out of the total collaboratively developed projects, approx. one-fifth of the projects (65,638 projects), observe development activities other than contribution to MF (refer 'Collaboratively Developed Projects' and 'Projects with Independently Developed Forks' in Table 15). An analysis of the temporal trends in column 'Projects with Independently Developed Forks' in Table 15 shows that every year projects with IDFs double in counts and represents a consistent fraction of projects in time. Noteworthy, we see a sudden dip in project counts that observe forking for the year 2012. This may be explained in terms of the insufficient development history for forking to occur at first place and for significant development activities to happen thereafter to examine the influence. Thus, forking is significant and is going to be more frequent in the future. This observation regarding the increasing relevance of forking for independent development is in agreement with the study by Robles *et al.* [147] who studied forking at project level.

4.1.5.1 *How forking influences the sustainability of the developer community participation in the original projects hosted on GitHub? Do projects ported to GitHub behave differently relative to projects created in GitHub?*

Out of the 5,424 projects with significant IDFs only 2,217 projects have significant developer community participation before and after forking for analysis (refer Total of 'Projects with IDFs' and 'Total Projects' in Table 15). This project count includes projects created and ported to GitHub. For identified projects, we compute changes in developer community participation in MF-PF pairs. Table 16 summarizes changes in the developer community participation as decrease, increase, and no effect for the MF-PF pairs created or ported to GitHub annually. Here, each entry summarizes the mean $\pm$ standard deviation of selected projects with 3, 6, 9, and 12 months of existence before and after forking.

In Table 16, we see that imported projects (1,881) are approx. three times in count with respect to the internally developed projects in GitHub (541). Approximately 58% of the internally developed projects in GitHub observe no effect in developer community participation after forking. Of the remaining, ≈33% and ≈9% of projects observe an increase and decrease in developer community participation respectively. Interestingly, more imported projects observe a decrease in developer community participation after forking (≈19%), while the fraction of the projects that observe no effect (≈50%) or observe an increase (≈30%) in developer community participation reduces proportionally.

Fig. 15, 16 and 17 show instances of three projects and the ways in which the sustainability of the developer community is influenced by forking. Here, the horizontal axis shows time-intervals measured monthly and the vertical axis shows developer community participation measured as commit counts per month. The dotted red line indicates the time of occurrence of forking. In Fig. 15, the developer community participation decreases after forking. In Fig. 17 developer community participation is not affected and in Fig. 16 developer community participation increases after forking.

Table 16: Classification of MF-PF pairs (or projects) developed *within* and *outside* GitHub as *decrease*, *increase*, and *no change* in sustainability of the developer community after forking. Each column entry is a summary (mean ± standard deviation) of MF-PF pairs having 3, 6, 9, and 12 months of existence before and after forking

Type	Year	Decrease	Increase	No Change	Total
Internal	2008	02±0	011±00	015±01	028±01
Internal	2009	18±0	156±02	084±06	258±08
Internal	2010	20±0	130±02	122±06	272±08
Internal	2011	28±1	148±06	210±20	387±27
Internal	2012	12±4	042±12	108±22	161±38
Projects Count		50	178	313	541
Percentage		≈9%	≈33%	≈58%	
Imported	2008	041±00	150±01	071±01	0216±02
Imported	2009	075±02	233±03	189±07	0496±12
Imported	2010	159±06	457±05	441±22	1057±33
Imported	2011	570±16	570±16	791±46	1650±78
Imported	2012	213±10	292±09	504±42	1009±60
Projects Count		365	569	947	1881
Percentage		≈20%	≈30%	≈50%	
All Projects Count		400	651	1166	2217
Percentage		≈18%	≈29%	≈53%	

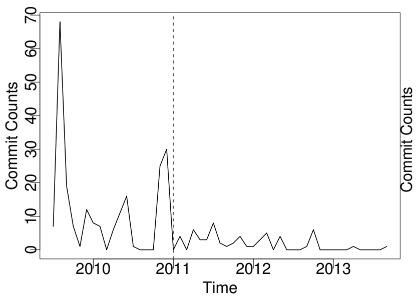


Figure 15: Decrease in participation

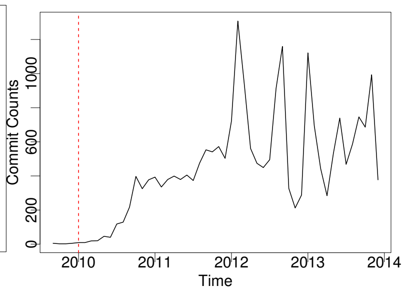


Figure 16: Increase in participation

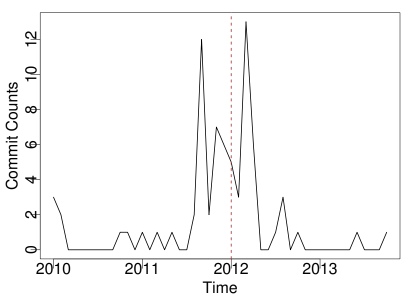


Figure 17: No effect in participation

Table 17: Influence of project characteristics on emergence of significant PF. Team size segments on contributors' count: Small < 11; 10 < Medium < 30; 30 < Large

	Small	Medium	Large
(Intercept)	$-8.62\ (0.10)^{***}$	$-6.00\ (0.20)^{***}$	$-5.36\ (0.09)^{***}$
projectAge	$0.47\ (0.02)^{***}$	$0.10\ (0.02)^{***}$	$0.11\ (0.02)^{***}$
mf_authorCount	$1.55\ (0.04)^{***}$	$0.75\ (0.07)^{***}$	$0.38\ (0.02)^{***}$
mf_followersCount	$-0.10\ (0.01)^{***}$	$-0.10\ (0.01)^{***}$	$-0.13\ (0.00)^{***}$
pf_followersCount	$0.05\ (0.01)^{***}$	$0.09\ (0.01)^{***}$	$0.18\ (0.00)^{***}$
mf_domainCompiler	$-0.05\ (0.13)$	$-0.18\ (0.13)$	$0.17\ (0.10)$
mf_domainDatabase	$0.19\ (0.10)$	$0.01\ (0.08)$	$0.66\ (0.09)^{***}$
mf_domainFramework	$-0.26\ (0.10)^{**}$	$0.02\ (0.08)$	$0.29\ (0.07)^{***}$
mf_domainlibrary	$-0.44\ (0.11)^{***}$	$-0.42\ (0.09)^{***}$	$0.07\ (0.08)$
mf_domainMiddleWare	$0.25\ (0.16)$	$0.45\ (0.12)^{***}$	$1.05\ (0.10)^{***}$
mf_domainOthers	$0.11\ (0.07)$	$0.14\ (0.05)^{*}$	$0.74\ (0.05)^{***}$
mf_watchersCount	$-0.27\ (0.01)^{***}$	$-0.24\ (0.01)^{***}$	$-0.27\ (0.01)^{***}$
AIC	25997.69	30564.25	42813.81
BIC	26145.54	30695.19	42947.10
Log Likelihood	−12986.85	−15270.12	−21394.91
Deviance	25973.69	30540.25	42789.81
Num. obs.	1656514	404926	492457

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

4.1.5.2 *How do project characteristics at the time of forking influence* forkability *of the original project?*

MATURITY With the rest of the characteristics being the same, in small projects, an increase in the maturity of a project by a year increases the odds of occurrence of a significant fork by 60%. However, for medium and large projects the increase in odds is 10% (refer projectAge in Table 17).

> Maturity of the MF is positively related to its forkability, with an attenuating effect as project size increases.

COMMUNITY SIZE In small projects, one unit increase in the contributors' count increases its forkability by 370%, which reduces to 110% and 40% for medium and large projects respectively (refer mf_authorCount in Table 17).

> An increase in developer community size of MF is positively related to its forkability with attenuating effect as project size increases.

OWNER One unit increase in the influence of the owner of MF is related with 10-12% decrease in odds of occurrence of forks for all sized projects (refer mf_followersCount in Table 17). However, one unit increase in the influence of the owner of PF is related with 5%, 9%, and 19% increase in the odds of occurrence of fork (refer pf_followersCount in Table 17).

> An increase in the influence of the owner of MF decreases the forkability of the MF.
> An increase in the influence of the owner of PF increases the forkability of MF.

DOMAIN We choose domain 'Application', with large project counts and developer base, as the base level to compare the impact of domain on significant forking. In Table 17, domain coefficients can be broadly categorized into three general categories. The first category includes those domains for which coefficients are statistically insignificant and the modeling procedure could not distinguish the impact from domain 'Application' (the base level). These domains behave like domain 'Application' or have wide variance. The remaining coefficients are either positive or negative. For domains with positive coefficients, the odds of occurrence of significant fork relative to domain 'Application' is positive. For instance, domain 'Middleware' in medium-sized projects and domain 'Database', 'Framework' and 'Middleware' in large-sized project. For those with negative coefficients, the odds of occurrence of significant fork are less relative to domain 'Application'. For instance, domain 'Framework' and 'Library' in small sized projects, and domain 'Library' in medium-sized projects. To sum up, domains influence the occurrence of significant fork. Domain 'Application' is more likely to observe significant forking relative to other domains in small-sized projects. It has mixed-effects on medium-sized projects and is less likely to observe significant forking relative to other domains in large-sized projects (refer domain in Table 17).

> The domain has significant, mixed-effects on the forkability of the MF.

POPULARITY An increase in the popularity of the MF by one unit decreases the odds of occurrence of a significant fork in all sized projects by approx. 21-23% (refer mf_watchersCount in Table 17).

> The more popular the MFs; less likely is its forkability.

4.1.5.3 *How do project characteristics at the time of forking influence the sustainability of the developer community participation?*

MATURITY In Table 18, the impact of project maturity (projectAge) is statistically significant for medium and large-sized teams. We observe that an increase in the maturity of MFs by a year reduces

Table 18: Influence of project characteristics on decrease in developer community participation in the original project. Team size segments on contributors' count: Small < 11; $10 <$ Medium < 30; $30 <$ Large

	Small	Medium	Large
(Intercept)	$-11.60\ (0.68)^{***}$	$-7.09\ (0.66)^{***}$	$-6.49\ (0.31)^{***}$
projectAge	$-0.11\ (0.12)$	$-0.27\ (0.07)^{***}$	$0.17\ (0.05)^{***}$
mf_authorCount	$1.42\ (0.28)^{***}$	$0.12\ (0.22)$	$-0.16\ (0.07)^{*}$
pf_authorCount	$2.64\ (0.14)^{***}$	$2.42\ (0.08)^{***}$	$1.70\ (0.05)^{***}$
mf_followersCount	$-0.19\ (0.04)^{***}$	$-0.09\ (0.02)^{***}$	$-0.18\ (0.02)^{***}$
pf_followersCount	$0.05\ (0.04)$	$0.01\ (0.02)$	$0.04\ (0.02)^{**}$
mf_domainCompiler	$0.28\ (0.66)$	$-1.73\ (0.62)^{**}$	$0.64\ (0.27)^{*}$
mf_domainDatabase	$-0.38\ (0.61)$	$0.13\ (0.24)$	$0.81\ (0.22)^{***}$
mf_domainFramework	$0.39\ (0.48)$	$0.21\ (0.23)$	$0.22\ (0.22)$
mf_domainlibrary	$0.04\ (0.52)$	$0.00\ (0.25)$	$-0.19\ (0.29)$
mf_domainMiddleWare	$0.31\ (0.78)$	$-1.06\ (0.53)^{*}$	$1.01\ (0.24)^{***}$
mf_domainOthers	$-0.26\ (0.36)$	$-0.60\ (0.17)^{***}$	$-0.08\ (0.16)$
mf_watchersCount	$-0.23\ (0.07)^{***}$	$-0.23\ (0.04)^{***}$	$-0.17\ (0.03)^{***}$
pf_watchersCount	$0.20\ (0.12)$	$0.15\ (0.08)$	$0.41\ (0.05)^{***}$
AIC	983.23	2708.90	4382.38
BIC	1155.71	2861.66	4537.88
Log Likelihood	-477.61	-1340.45	-2177.19
Deviance	955.23	2680.90	4354.38
Num. obs.	1656514	404926	492457

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

the odds of decline in the developer community participation by 23% in medium size projects and increases the odds of decline by 18% in large size projects (refer projectAge in Table 18).

> An increase in the maturity of the MF combats decline in medium sized projects and supports decline in large sized projects.

community size In small projects, an increase in the community size of MF by one unit increases the chances of decline in the developer community participation by 313%. In large projects, an increase in the community size of MF by one unit decreases the odds of decline in the developer community participation by 15%.

An increase in the community size of PF by one unit increases the odds of decline in the developer community participation by 13, 10, and 5 times.

> An increase in the community size of MF increases the chances of decline in developer community participation in small projects.
>
> An increase in the community size of MF decreases the chances of decline in developer community participation in large projects.
>
> An increase in the community size of PF increases the chances of decline in developer community participation in the original project.

owner In Table 18, an increase in the influence of the owner (refer mf_followersCount) decreases the odds of decline in the developer community participation by 17%, 8%, and 16% for small, medium and large projects respectively.

In large projects, an increase in the influence of the owner of the PF increases the odds of decline in the developer community participation by 4%.

> An increase in the influence of the owner of the MF decreases the chances of decline in developer community participation in the MF.
>
> An increase in the influence of the owner of the PF increases the chances of decline in developer community participation in the MF.

DOMAIN The influence of domain is statistically insignificant for small-sized projects. For medium-sized projects, the odds of decline of developer community participation decrease by 82% for domain 'Complier' and 40% for domain 'Middleware' relative to domain 'Application' (refer Table 18). However, for large projects, odds of decline in developer community participation relative to 'Application' increases by 89%, 124%, and 174% for domains 'Compiler', 'Database', and 'Middleware' respectively. In short, projects in domain 'Application' are less likely to observe a decline as project size increases.

The domain has significant, mixed-effects on the decline in developer community participation.

POPULARITY In Table 18, increase in the popularity of the original project (refer mf_watchersCount) decreases the odds of decline in the developer community participation by 20%, 20% and 15% in small, medium and large projects respectively.

In large projects, an increase in the popularity of the PF increases the odds of decline in the developer community participation by 51%.

An increase in the popularity of the MF decreases the chances of decline in developer community participation in the original project after forking.

An increase in the popularity of the PF increases the chances of decline in developer community participation in the original project after forking.

4.1.6 *Threats to Validity*

We try to address concerns related to Git[155] or GitHub[153] by removing projects with too few commits and also one-person projects. However, our attempts to quantitatively measure the influence of forking are subjected to limitations.

- *Internal Validity*
  - *Data Accuracy*: GHTorrent data provide a fair estimate of activities and events in a project and has been used in various studies. However, projects where history is rewritten may provide an underestimation of the developer community participation and classification of forks.
  - *Classification of Forks*: Some forks start as contributing and then grow to become independent or vice-versa. Since we do not include the temporal aspect here, we may end up classifying both types of projects as contributing.
  - *Collaborative Development Model*: We study collaboration through Fork & Pull model. So, projects which use a mix of Fork & Pull and shared repository model are not modeled completely.

- *External Validity* We present a case study of projects hosted on GitHub. While GitHub represents a diverse set of projects, for generalization, similar experiments must be conducted on other coding sites like BitBucket, etc.

4.1.7 *Implications*

Forking is significant and is going to be more relevant in the future. An analysis of more than 2,000 OSS projects shows that forking alters developer community participation in approximately half of the projects. Interestingly, it increases developer community participation in more projects than it decreases developer community participation. Yet, the decrease observed influences a significant fraction of the projects. Also, the effect is more pronounced in imported projects than internally developed projects. Further, to understand the cause of the decline in the sustainability of the developer community participation, we study the influence of various project characteristics at the time of forking. We found that an increase in the maturity of the MF, its community size and the influence of the owner of the PF increases the forkability of the original project. An increase in the influence of the owner of MF and the popularity of the MF decreases the forkability. The domain has a significant, mixed-effect on the forkability. We see that an increase in the maturity of MF decreases the chances of decline in the developer community participation in medium sized projects and increases the chances of decline in large sized projects. Similarly, an increase in the community size of MF increases and then decreases the chances of decline in developer community participation as project size increases. Besides this, an increase in the community size of PF, the influence of its owner, and its popularity increases the chances of decline in the developer community participation in the original project. An increase in the influence of the owner of MF and its popularity decreases the chances of decline in the developer community participation in the original project. Among others, these observations explain why fork LibreOffice has no effect on the sustainability of the developer community participation in OpenOffice. It is important to note here that some factors have effects both ways. For instance, an increase in the maturity of the MF increases the forkability, while also decreasing the chances of decline in developer community participation of MF.

4.2 PERSONALITY TRAITS AND CONTEXT OF SOFTWARE DEVELOPMENT

4.2.1 *Introduction*

Studies in Psychology suggest that behavior is best predicted from a comprehensive understanding of the person, the situation and the interactions between the person and the situation [156]. In this study, we intend to empirically explain the behavior of contributors by investigating their personality profiles - unique to them, and the context of software development. We believe that an understanding of the behavior of contributors will help comprehend the true nature of online distributed software development. This understanding of contributors' behavior can be used to facilitate software development process thereby improving project performance. For instance, personality traits can be used to explain productivity [157], withdrawal behavior [158], etc.

Studies on commercial software development projects [159] [160] [161] [162] and online platforms like StackOverflow [163] and Twitter [164] have shown the importance of personality traits in explaining behavior. A preliminary study on an OSS project also acknowledged the inferential ability of personality traits in explaining contributors' joining and leaving behavior [158].

In this study, we measure the personality profiles of contributors, as evident from their language use [165], in discussions on software development. Personality profiles of contributors are described through five traits namely Openness to Experience, Conscientiousness, Extraversion, Agreeableness and Neuroticism [166]. Openness to Experience is related to being insightful and open to new ideas [157]. Conscientiousness is the preference for order and goal-directed work [157]. Extraversion describes individuals' desire to seek company and derives stimulation from the external world [157]. Agreeable people are cooperative, compassionate and sensitive to others [157]. Neuroticism is related to the tendency to express negative emotions and anger [157].

We empirically analyze the relationships of the personality traits of contributors with contributions and context of software development in GitHub. Specifically, we study differences in the personality traits of sub-communities of contributors with different levels of contributions and project membership status. Also, we study differences in the personality traits of individuals when they contribute in different contexts. Particularly, we look for changes in the personality traits with time, type of contribution, role and project membership. We leverage the software development project data in GitHub to provide answers to the following project-level and contributor-level research questions:

Project-level

RQ1) Do personality types vary with the contribution?

Contributors who contribute more score high on Openness to Experience, Conscientiousness, and Extraversion. Agreeableness decreases as contribution increases. Contributors with extreme (high and low) levels of contributions are more Neurotic compared to the contributors with medium-level of contributions.

RQ2) Do personality types vary with project membership?

There are no differences in Openness to Experience and Neuroticism based on the membership of contributors in the projects. Project members are more Conscientious and Extrovert and less Agreeable compared to non-project members.

Contributor-level

RQ3) Do personality traits of contributors evolve with time?

There are no changes in Openness to Experience and Neuroticism. Active contributors evolve as more Conscientious, more Extrovert and less Agreeable with time.

RQ4) Do contributors show different personality traits for different contribution types?

There are no differences in Openness to Experience based on the contribution type. Active contributors are most Conscientious, Extrovert and Neurotic while discussing pull requests and most Agreeable while discussing issues.

RQ5) Do contributors show different personality traits for different roles?

As reporters, highly active contributors are more Open to Experience, Conscientious and Neurotic compared to when they act as reviewers. Contributors are more extrovert as reviewers. There is no change in Agreeableness.

RQ6) Do contributors show different personality traits based on project membership?

Active contributors are more Conscientious, Agreeable and Neurotic then when they are not the member of the project. Contributors are more extrovert when they are the member of the project. There is no change in Openness to Experience. It is important to note here that in RQ2, we compare the personality profiles of sub-communities of contributors who are project members against non-project members. While in RQ6, we compare the personality profiles of contributors in projects where they are project members against their personality profiles in projects where they are not project members.

We presented several results that match our expectations, thereby demonstrating the inferential power of the personality traits in explaining the behavior in various contexts of software development in GitHub. We see this study as a first step towards comprehending the intricacies of a team by creating a better understanding of contributors' behavior.

4.2.2 *Background and Related Work*

4.2.2.1 *Personality Theories and Tools*

Personality theories use validated tests to capture individuals' personality profiles. The two most popular personality models are the Big Five [167] (with a variation called the Five Factor Model [168]) and Myers-Briggs Type Indicator (MBTI) [169]. The Big Five personality model is developed on the underlying position that "personality is encoded in natural language, and differences in personality may become apparent through language use" [170]. In contrast, MBTI model was developed on the stance that "individuals evolve psychologically through experiences and this evolution shapes individuals' behavior along different psychological types" [171]. Studies say that MBTI is suitable for assessing individual's self-awareness as against its common use for explaining performance [172]. MBTI has reliability and validity issues too [173]. We use the Big Five model to measure personality traits in this study. The Big Five personality profiles have been correlated with individuals' language use [174] - the phenomenon under consideration in this work.

Individual's linguistic style is quite stable [175] [176]. Personality traits can be detected in individual's interactions, even if they occur in textual settings [175]. There are text analytics programs that accurately links language characteristics to personality traits with a modest but reliable effect [175] [176]. For instance, use of pronouns or articles may not reveal which objects or events a person is talking about, but it provides a sense of the person's general approach to the world.

We use LIWC software tool [177] to measure personality traits in terms of the language used. This tool captures over 86% of the words used during conversations (around 4500 words) [177]. The reliability and the validity of the tool have been extensively evaluated by the studies on various platforms such as Facebook [178], Twitter [164] [179], and StackOverflow [180] [163]. We chose LIWC tool for analysis because comments on GitHub are short and informal, similar to Facebook and Twitter - which used LIWC tool. Further, existing studies on software development provide useful discoveries to encourage the systematic application of linguistic analysis techniques to study developers' communications [158][157].

4.2.2.2 *Textual Communication, Personality Traits and Software Development*

Communication archives in software development projects are shown to contain significant information about the software systems they discuss [181]. These archives are also used to infer personality profiles which have a significant influence on people's job behavior [182].

In commercial software development projects, the personality of project managers influences the success of projects [162]. Attitude and behavior of the software development team vary with the nature of task performed [159]. Further, high levels of organizational and interpersonal skills are found useful for those operating in distributed settings, and that personality diversity boosts team performance [157].

Preliminary analyses of the personality traits of four developers on a large, popular, open source software project show promise in understanding developers' joining and leaving pattern [158]. Differences in the personality traits of contributors were also observed [158]. Building on the existing studies, here we explore the inferential power of the personality traits in explaining the behavior of contributors in various contexts of software development in GitHub.

4.2.3 *Methodology*

For RQ1 and RQ2, we classify the contributors in projects into groups based on their levels of contributions and project membership respectively. Similarly, for RQ3 to RQ6, we group the contributions of contributors based on time of participation, contribution type, role and project membership respectively. We compute the personality traits of contributors in each group and study their relationships with the above-mentioned factors. The step-by-step procedure used in this study is summarized in Figure 18. All the data and procedures used here is made publicly available for replication[13].

4.2.3.1 *Data Preparation*

We downloaded the GitHub projects' data publicly available by GHTorrent[14] [154]. The dataset has development history of more than six years (October 2007 to August 2014). The selection criteria of projects and contributors is mentioned below:

4.2.3.2 *Project Selection Criteria*

We selected 243 actively discussed projects. These projects are the union of top 100 projects selected in terms of comments count to discuss issues, pull requests and commits. The selected projects have 119,638 contributors who participated at least once in discussions. Refer step 1, part 1 (Identify target audience) of Figure 18.

4.2.3.3 *Contributor Selection Criteria*

We selected 423 active contributors in discussions. These contributors wrote at least 100 comments during discussions on issues, pull requests and commits separately. The participation of these selected contributors is not limited to the projects selected in sub-section 4.2.3.2. Refer step 1, part 2 (Identify target audience) of Figure 18.

13 https://sites.google.com/site/ayushirastogi1989india/research/personalitytraits-data
14 http://ghtorrent.org/downloads.html

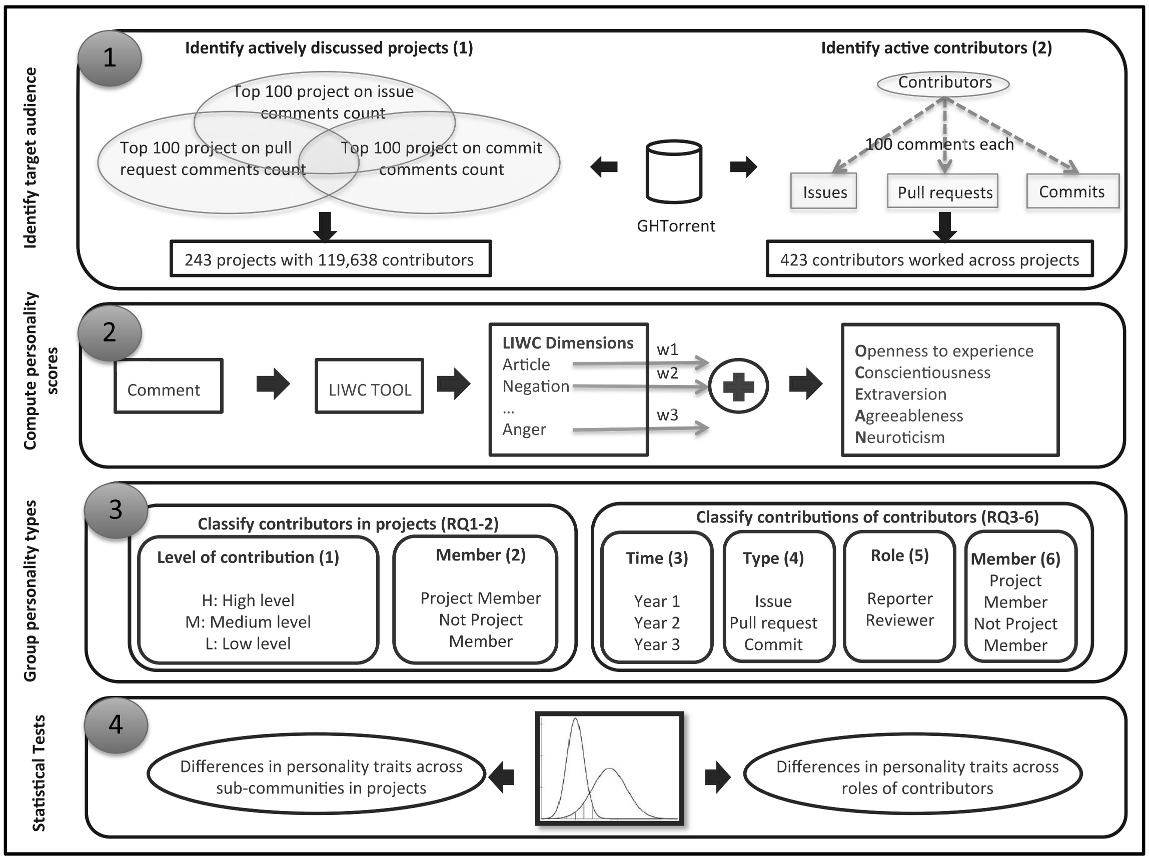

Identify target audience
1
Identify actively discussed projects (1)
Top 100 project on issue comments count
Top 100 project on pull request comments count
Top 100 project on commit comments count
243 projects with 119,638 contributors
GHTorrent
Identify active contributors (2)
Contributors
100 comments each
Issues
Pull requests
Commits
423 contributors worked across projects
Compute personality scores
2
Comment
LIWC TOOL
LIWC Dimensions
Article
Negation
...
Anger
w1
w2
w3
Openness to experience
Conscientiousness
Extraversion
Agreeableness
Neuroticism
Group personality types
3
Classify contributors in projects (RQ1-2)
Level of contribution (1)
H: High level
M: Medium level
L: Low level
Member (2)
Project Member
Not Project Member
Classify contributions of contributors (RQ3-6)
Time (3)
Year 1
Year 2
Year 3
Type (4)
Issue
Pull request
Commit
Role (5)
Reporter
Reviewer
Member (6)
Project Member
Not Project Member
Statistical Tests
4
Differences in personality traits across sub-communities in projects
Differences in personality traits across roles of contributors

Figure 18: *Research Method* [Step 1] Identify projects and contributors for the study. [Step 2] Compute personality traits score from discussions. [Step 3] Group contributors based on contribution and project membership. Group contribution of contributors based on time, type, role and project membership. [Step 4] Conduct statistical analyses to test for significance of the differences in personality traits across groups.

4.2.3.4 *Measure*

personality traits Yarkoni [175] measured personality traits of individuals using LIWC tool. LIWC tool uses a dictionary for text analysis. The default LIWC 2007 dictionary is made up of more than 4,500 words and word stems. Each word or word stem is defined in one or more word categories or sub-dictionaries. These categories or LIWC dimensions are arranged hierarchically. For instance, 'psychological process' consists of 'social processes', 'affective processes', 'cognitive processes', etc.

Yarkoni measured personality traits as the weighted sum of LIWC dimensions. Here, the weight was the correlation coefficient between the LIWC dimension and the personality trait. Thus, LIWC dimension which has a high correlation with the personality trait is given more weight. The measure of Neuroticism proposed by Yarkoni [175] is:

(0.12 × FirstPersonSingular) + (0.10 × FirstPerson) − (0.15 × SecondPerson) + (0.11 × Negation) − (0.11 × Article) − ~~(0.08 × Optimism)~~ + (0.16 × NegativeEmotion) + (0.17 × Anxiety/Fear) + (0.13 × Anger) + (0.10 × Sadness) + (0.13 × CognitiveProcess) + (0.11 × Causation) + (0.13 × Discrepancy) + ~~(0.09 × Inhibition)~~ + (0.12 × Tentative) + (0.13 × Certainty) + (0.10 × Feeling) − ~~(0.08 × OtherReferences)~~ − (0.08 × Friends) − (0.09 × Spaces) − ~~(0.10 × Up)~~ + (0.10 × Exclusive) + (0.10 × Sleeping) + (0.11 × SwearWords)

Here, deleted terms were seen to correlate with personality traits when using LIWC 2001 dictionary, but were removed from LIWC 2007 due to low base rate [176]. Similarly, Yarkoni [175] proposed measures for the other four personality traits. In this study, we use the measures of Personality Traits proposed by Yarkoni [175]. It is important to note that the measures of personality traits are composite and hence the absolute values are meaningless. However, comparing the relative scores helps determine differences in the personalities [158].

We apply LIWC tool to each comment and calculate the values of the "Big Five Personality Traits". Refer step 2 (Compute personality scores) of Figure 18. For instance,

Comment 1: "Yep, you are right, the problem is that we don't know if more headers have been added. Another option is to create a 'refresh' method and call it from outside every time a new header is created/removed. Makes sense for me."
Comment 2: "Why did these need to change? the fact that it didn't break any tests means we have some untested and unverified cases here as well."

Using the formula for Neuroticism mentioned in equation 4.2.3.4, comment 1 has a lower neuroticism score (5) relative to comment 2 (8).

CONTRIBUTION Contributors participate (in issues) by reporting issues or by helping get it fixed. Similarly, contributors participate in pull requests and commits. We measure contribution as the count of issues, pull requests and commits on which the contributor participated in any form and to any degree.

4.2.3.5 *Classify Contributors in Projects*

CONTRIBUTION (RQ1) The distribution of contributors based on their contributions follows a skewed distribution [183]. Therefore, similar to other studies [163], we classify contributors into three classes as follows: 1% of contributors as high contribution contributors (H), 10% of contributors as medium contribution contributors (M) and the remaining 89% of contributors as low contribution contributors (L). We study contributions of individuals with at least 10 comments to have a reliable estimate of their personality traits. Refer step 3, part 1, RQ1 (Classify contributors in projects) in Figure 18.

PROJECT MEMBERSHIP (RQ2) We classify contributors in each project into two classes based on whether they are a member of the project or not. Contributors with commit access are members of the project. Refer step 3, part 1, RQ2 (Classify contributors in projects) in Figure 18.

4.2.3.6 *Classify Contribution of Contributors*

TIME (RQ3) We classify contributions of contributors based on the year of participation. We examine participation in three consecutive years. Refer step 3, part 2, RQ3 (Classify contributions of contributors) in Figure 18.

CONTRIBUTION TYPE (RQ4) We classify contributions of contributors based on participation in issues, pull requests and commits. Refer step 3, part 2, RQ4 (Classify contributions of contributors) in Figure 18.

ROLE (RQ5) We classify contributions of contributors based on their role as a reporter or a reviewer. Refer step 3, part 2, RQ5 (Classify contributions of contributors) in Figure 18.

PROJECT MEMBERSHIP (RQ6) We classify contributions of contributors based on whether they are a member of the project or not. Similar to RQ2, contributors with commit access are members of the project. Refer step 3, part 2, RQ6 (Classify contributions of contributors) in Figure 18.

4.2.3.7 *Statistical Tests*

We conduct statistical tests to examine differences in personality traits in two cases: 1) examine differences in personality traits across the sub-communities of projects (RQ1 and RQ2) and 2) examine differences in personality traits across the roles of contributors (RQ3-RQ6). Refer step 4 in Figure 18.

To compare the personality traits of contributors in two contexts (as in RQ3, RQ5, and RQ6), we conducted pairwise t-test and reported the effect size using Cohen's d. For RQ2, where the observations are independent, we conducted t-test followed by Cohen's d. For the five personality traits, we report mean and standard deviation of group 1 (M1; SD1) and group 2 (M2; SD2). We report t-statistics (t) with the degree of freedom (d), p-value (p) and Cohen's d (d). We also report differences in mean. The differences are significant for $p<0.001$.

To compare personality traits across more than two groups (as in RQ1 and RQ4), we conducted ANOVA to compare the means of the distribution. If significant differences were observed, we conducted Tukey's HSD to test for significant differences between all pairs. We report effect size using

Table 19: (RQ1) Differences in the personality traits of contributors across projects with high (H), medium (M), and low (L) levels of contributions

Personality Trait	ANOVA			Tukey's HSD					
	F(df, dferror)	p-value	η^2	M-H		L-H		L-M	
				diff	p-value	diff	p-value	diff	p-value
O	F(2,18906)=166.8	0.000	0.017	-0.16	0.09	-0.84	0.000	-0.68	0.000
C	F(2,18906)=421.4	0.000	0.042	-0.65	0.000	-1.21	0.000	-0.55	0.000
E	F(2,18906)=78.48	0.000	0.008	-0.45	0.000	-0.68	0.000	-0.23	0.000
A	F(2,18906)=121.7	0.000	0.012	0.09	0.01	0.34	0.000	0.24	0.000
N	F(2,18906)=13.22	0.000	0.001	-0.20	0.000	-0.07	0.20	0.12	0.000

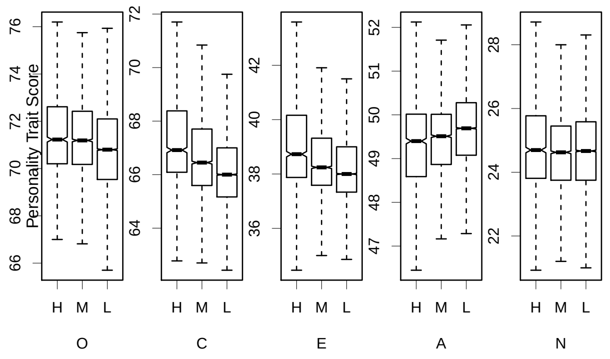


Figure 19: Differences in the personality traits of contributors across projects with High (H), Medium (M) and Low (L) levels of contributions

eta squared (η^2). For the five personality traits, we report F-statistics along with the degree of freedom and degree of error as F(df, dferror). We report p-values (p) and eta square (η^2). We report the results of Tukey's HSD as the differences in mean (diff) for all pairs and their p-values (p-value). The differences are significant for $p<0.001$.

We show differences in the personality traits across groups using notched boxplot (for instance, refer Figure 19). The vertical axis on the boxplot shows the relative personality traits score and the horizontal axis shows groups. Each figure presents the results of the five personality traits (Openness to Experience, Conscientiousness, Extraversion, Agreeableness, Neuroticism). The interpretation of the figure is that that if the notches of the two groups do not overlap, there are statistically significant differences in the personality traits of the groups. All the tests are conducted using R language [184].

4.2.4 *Results*

RQ1: DO PERSONALITY TYPES VARY WITH CONTRIBUTION? We compared the personality traits of contributors with different (high, low, and medium) levels of contributions. There is a significant effect of the level of contribution on the Openness to Experience of contributors F(2,18906)=166.8, p=0.000, η^2=0.017. There is no difference in the Openness to Experience between high and medium levels of contributors diff=-0.16, p=0.09. Openness to Experience decreases with the decrease in the level of contribution. There is a significant effect of the level of contribution on the Conscientiousness of contributors F(2,18906)=421.4, p=0.000, η^2=0.042. Conscientiousness decreases with the decrease in the levels of contributions. There is a significant effect of the level of contribution on the Extraversion of contributors F(2,18906)=78.48, p=0.000, η^2=0.008. Extraversion decreases with the decrease in the levels of contributions. There is a significant effect of the level of contribution on the Agreeableness of contributors F(2,18906)=121.7, p=0.000, η^2=0.012. Agreeableness is similar for high and medium contribution contributors diff=0.09, p=0.01 and decreases for low contribution contributions. There is a significant effect of the level of contribution on the Neuroticism of contributors F(2,18906)=13.22, p=0.000, η^2=0.001. Neuroticism is high for high and low contribution contributors diff=-0.07, p=0.20 and decreases for medium contribution contributors. Refer Table 19 and Figure 19 for the summary of the observations.

RQ2: DO PERSONALITY TYPES VARY WITH PROJECT MEMBERSHIP? We compared the personality traits of contributors who were the member of the selected projects against the personality traits

Table 20: (RQ2) Differences in the personality traits of contributors across projects based on whether they are a member of the project (PM) or not project member (NPM)

Groups	Personality Trait	M1	SD1	M2	SD2	df	t	p	d	Mean of differences
	O	71.3	2.2	71.1	2.5	3781	2.66	0.007	0.05	-
	C	67.1	1.7	66.2	1.6	3503	24.06	0.000	0.50	0.86 ↑
PM and NPM	E	39.6	2.5	38.4	1.9	3136	22.06	0.000	0.55	1.14 ↑
	A	49.3	1.1	49.5	1.0	3341	-7.05	0.000	0.15	0.17 ↓
	N	24.7	1.6	24.6	1.5	3426	2.59	0.009	0.05	-

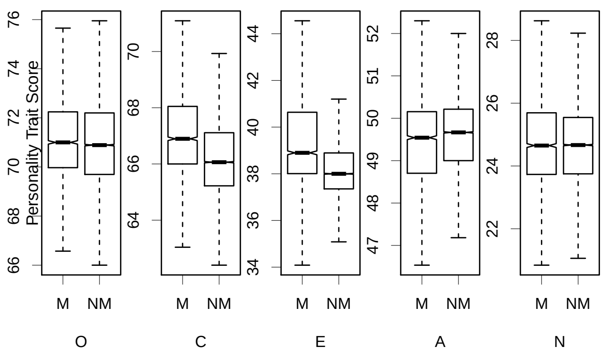


Figure 20: Differences in the personality traits of contributors across projects based on whether they are a member of the project (M) or not (NM)

of contributors who were not the member of the selected projects. We find that there is no significant difference in the Openness to Experience of project members (M=71.3, SD=2.2) from non-project members (M=71.1, SD=2.5), t(3781)=2.66, p=0.007, d=0.05. There is a significant difference in Conscientiousness of project members (M=67.1, SD=1.7) from non-project members (M=66.2, SD=1.6), t(3503)=24.06, p=0.000, d=0.50. Project members are high on Conscientiousness by 0.86 compared to the non-project members. There is a significant difference in the the Extraversion of project members (M=39.6, SD=2.5) from non-project members (M=38.4, SD=1.9), t(3136)=22.06, p=0.000, d=0.55. Extraversion increases by 1.14 from non-project members to project members. There is a significant difference in the Agreeableness of project members (M=49.3, SD=1.1) from non-project members (M=49.5, SD=1.0), t(3341)=-7.05, p=0.000, d=0.15. Agreeableness decreases by 0.17 from non-project members to project members. There is no significant difference in the Neuroticism of project members (M=24.7, SD=1.6) from non-project members (M=24.6, SD=1.5), t(3426)=2.59, p=0.009, d=0.05. Refer Table 20 and Figure 20 for the summary of the observations.

RQ3: DO PERSONALITY TRAITS OF CONTRIBUTORS EVOLVE WITH TIME? We compared the personality traits of contributors during the first year and the second year of their participation. Out of the selected 423 contributors, 413 contributors participated for two consecutive years. We find that there is no significant difference in the Openness to Experience from year 1 (M=70.9, SD=3.3) to year 2 (M=71.1, SD=1.9), paired t(412)=-0.93, p=0.346, d=0.05. There is a significant difference in the Conscientiousness from year 1 (M=66.7, SD=2.2) to year 2 (M=67.0, SD=1.5), paired t(412)=-3.21, p=0.001,

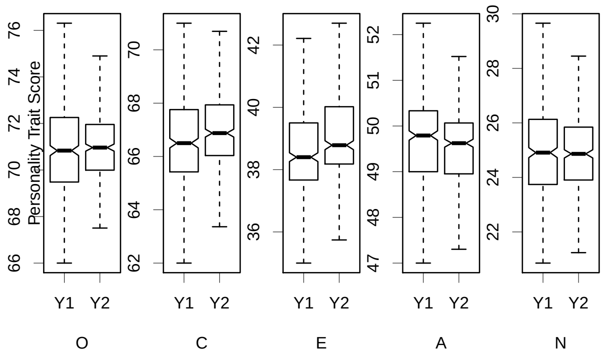


Figure 21: Differences in the personality traits of the contributors during the first year (Y1) and the second year (Y2) of their participation

Table 21: (RQ3,5,6) Differences in the personality traits of contributors with time (Year 1, Year 2 and Year 3), role (Reporter Rep and Reviewer Rev) and project membership (Project Member PM and Non Project Member NPM)

Paired Groups	Personality Trait	M1	SD1	M2	SD2	df	t	p	d	Mean of differences
Year 1 and Year 2	O	70.9	3.3	71.1	1.9	412	-0.93	0.346	0.05	-
	C	66.7	2.2	67.0	1.5	412	-3.21	0.001	0.17	0.34 ↑
	E	38.8	2.5	39.3	1.9	412	-3.51	0.000	0.20	0.45 ↑
	A	49.6	1.3	49.4	0.9	412	3.64	0.000	0.19	0.22 ↓
	N	25.0	2.5	24.9	1.6	412	0.72	0.467	0.04	-
Year 1 and Year 3	O	70.8	3.4	71.0	1.8	366	-0.98	0.326	0.06	-
	C	66.5	2.2	67.1	1.4	366	-4.3	0.000	0.27	0.52 ↑
	E	38.6	2.4	39.6	2.1	366	-6.2	0.000	0.42	0.99 ↑
	A	49.7	1.3	49.4	0.9	366	5.97	0.000	0.37	0.45 ↓
	N	24.9	2.6	25.2	1.3	366	-1.9	0.053	0.12	-
Rep and Rev	O	71.3	1.9	71.1	1.6	422	3.49	0.000	0.11	0.21 ↓
	C	68.3	1.6	67.1	1.2	422	21.12	0.000	0.83	1.23 ↓
	E	39.4	2.2	40.0	1.9	422	-4.66	0.000	0.27	0.58 ↑
	A	49.1	0.9	49.1	0.9	422	-1.02	0.304	0.04	-
	N	26.4	1.5	25.0	1.1	422	21.34	0.000	0.99	1.39 ↓
PM and NPM	O	71.1	1.8	71.2	1.7	396	-0.61	0.535	0.02	-
	C	67.3	1.5	67.6	1.5	396	-3.95	0.000	0.17	0.27 ↑
	E	40.0	2.0	39.2	1.6	396	8.29	0.000	0.46	0.84 ↓
	A	49.1	1.0	49.3	0.8	396	-4.90	0.000	0.20	0.19 ↑
	N	25.2	1.5	25.6	1.4	396	-5.91	0.000	0.28	0.42 ↑

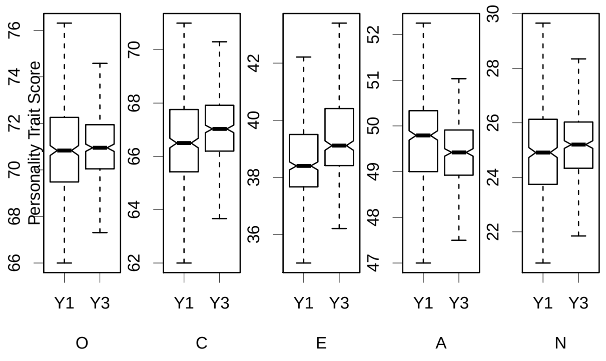


Figure 22: Differences in the personality traits of contributors during the first year (Y1) and the third year (Y3) of their participation

d=0.17. Conscientiousness increases by 0.34 from year 1 to year 2. There is a significant difference in the Extraversion from year 1 (M=38.8, SD=2.5) to year 2 (M=39.3, SD=1.9), paired t(412)=-3.51, p=0.000, d=0.20. Extraversion increases by 0.45 from year 1 to year 2. There is a significant difference in the Agreeableness from year 1 (M=49.6, SD=1.3) to year 2 (M=49.4, SD=0.9), paired t(412)=3.64, p=0.000, d=0.19. Agreeableness decreases by 0.22 from year 1 to year 2. There is no significant difference in the Neuroticism from year 1 (M=25.0, SD=2.5) to year 2 (M=24.9, SD=1.6), paired t(412)=0.72, p=0.467, d=0.04. Refer Table 21 and Figure 21 for the summary of the observations.

We also compared the personality traits of contributors during the first year and the third year of their participation. Out of the selected 423 contributors, 367 contributors participated for three consecutive years. We find that there is no significant difference in the Openness to Experience from year 1 (M=70.8, SD=3.4) to year 3 (M=71.0, SD=1.8), paired t(366)=-0.98, p=0.326, d=0.06. There is a significant difference in the Conscientiousness from year 1 (M=66.5, SD=2.2) to year 3 (M=67.1, SD=1.4), paired t(366)=-4.3, p=0.000, d=0.27. Conscientiousness increases by 0.52 from year 1 to year 3. There is a significant difference in the Extraversion from year 1 (M=38.6, SD=2.4) to year 3 (M=39.6, SD=2.1), paired t(366)=-6.2, p=0.000, d=0.42. Extraversion increases by 0.99 from year 1 to year 3. There is a significant difference in the Agreeableness from year 1 (M=49.7, SD=1.3) to year 3 (M=49.4, SD=0.9), paired t(366)=5.97, p=0.000, d=0.37. Agreeableness decreases by 0.45 from year 1 to year 3. There is no significant difference in the Neuroticism from year 1 (M=24.9, SD=2.6) to year 3 (M=25.2,

Table 22: (RQ4) Differences in the personality traits of contributors while discussing issues (I), pull requests (P) and commits (C)

Personality Trait	ANOVA			Tukey's HSD					
	F(df, dferror)	p-value	η^2	P-I		C-I		C-P	
				diff	p-value	diff	p-value	diff	p-value
O	F(2,1266)=3.3	0.036	0.005	0.23	0.18	0.33	0.030	0.10	0.72
C	F(2,1266)=2044	0.000	0.76	4.60	0.000	3.93	0.000	-0.67	0.000
E	F(2,1266)=14392	0.000	0.95	9.10	0.000	-0.24	0.000	-9.34	0.000
A	F(2,1266)=2917	0.000	0.82	-3.59	0.000	-0.39	0.000	3.19	0.000
N	F(2,1266)=2129	0.000	0.77	4.91	0.000	4.45	0.000	-0.45	0.000

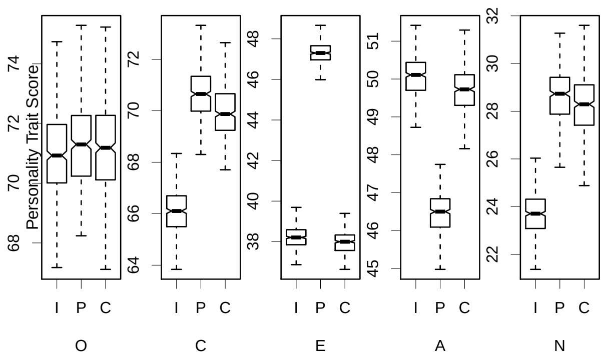


Figure 23: Differences in the personality traits of contributors while discussing issues (I), pull requests (P) and commits (C)

SD=1.3), paired t(366)=-1.9, p=0.053, d=0.12. Refer Table 21 and Figure 22 for the summary of the observations.

rq4: do contributors show different personality traits for different contribution types? We compared the personality traits of contributors when discussing issues, pull requests and commits. There is no effect of the contribution type on the Openness to Experience of contributors F(2,1266)=3.3, p=0.036, η^2=0.005. There is a significant effect of the contribution type on the Conscientiousness of contributors F(2,1266)=2044, p=0.000, η^2=0.76. Conscientiousness is highest when working on pull requests and decreases for commits and issues sequentially. There is a significant effect of the contribution type on the Extraversion of contributors F(2,1266)=14392, p=0.000, η^2=0.95. Extraversion is highest when working on pull requests and decreases while working on issues and commits sequentially. There is a significant effect of the contribution type on the Agreeableness of contributors F(2,1266)=2917, p=0.000, η^2=0.82. Agreeableness is highest when working on issues and decreases while working on commits and pull requests sequentially. There is a significant effect of the contribution type on the Neuroticism of contributors F(2,1266)=2129, p=0.000, η^2=0.77. Neuroticism is highest when working on pull requests and decreases while working on commits and issues sequentially. Refer Table 22 and Figure 23 for the summary of the observations.

rq5: do contributors show different personality traits for different roles? We compared the personality traits of contributors for their roles of reporter and reviewer. We find that there is a significant difference in the Openness to Experience from the role reporter (M=71.3, SD=1.9) to the role reviewer (M=71.1, SD=1.6), paired t(422)=3.49, p=0.000, d=0.11. Openness to Experience decreases by 0.21 for the role of reviewer compared to the role of reporter. There is a significant difference in the Conscientiousness from the role reporter (M=68.3, SD=1.6) to the role reviewer (M=67.1, SD=1.2), paired t(422)=21.12, p=0.000, d=0.83. Conscientiousness decreases by 1.23 from the role reporter to the role reviewer. There is a significant difference in the Extraversion from the role reporter (M=39.4, SD=2.2) to the role reviewer (M=40.0, SD=1.9), paired t(422)=-4.66, p=0.000, d=0.58. Extraversion increases by 0.58 from the role reporter to the role reviewer. There is no significant difference in the Agreeableness from the role reporter (M=49.1, SD=0.9) to the role reviewer (M=49.1, SD=0.9), paired t(422)=-1.02, p=0.304, d=0.04. There is a significant difference in the Neuroticism from the role reporter (M=26.4, SD=1.5) to the role reviewer (M=25.0, SD=1.1), paired t(422)=21.34, p=0.000, d=0.99.

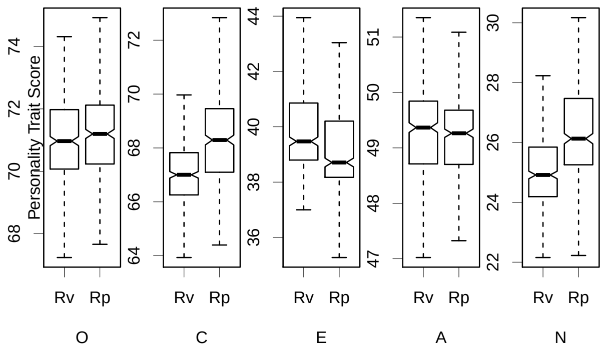


Figure 24: Differences in the personality traits of contributors when they are reporter (Rp) versus when they are reviewer (Rv)

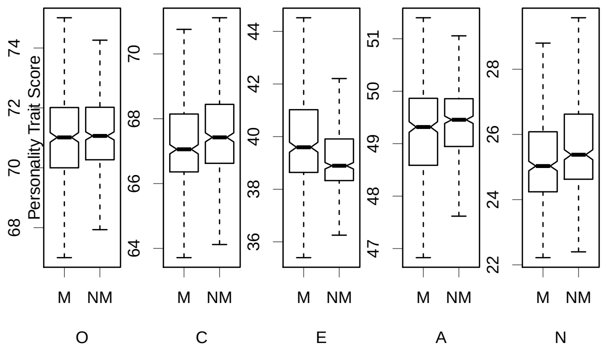


Figure 25: Differences in the personality traits of the contributors based on whether they are a member of the project (M) or not (NM)

Neuroticism increases by 1.39 from the role reporter to the role reviewer. Refer Table 21 and Figure 24 for the summary of the observations.

RQ6: DO CONTRIBUTORS SHOW DIFFERENT PERSONALITY TRAITS BASED ON PROJECT MEMBERSHIP? We compared the personality traits of contributors when they played project member and non-project member. Out of the 423 contributors, only 397 contributors played project member in some projects and non-project member in other projects. We find that there is no significant difference in the Openness to Experience when contributors are project members (M=71.1, SD=1.8) compared to when they are non-project member (M=71.2, SD=1.7), paired t(396)=-0.61, p=0.535, d=0.02. There is a significant difference in the Conscientiousness when contributors are project members (M=67.3, SD=1.5) compared to when they are non-project members (M=67.6, SD=1.5), paired t(396)=-3.95, p=0.000, d=0.17. Conscientiousness increases by 0.27 for non-project members compared to project members. There is a significant difference in the Extraversion when contributors are project members (M=40.0, SD=2.0) compared to when they are non-project members (M=39.2, SD=1.6), paired t(396)=8.29, p=0.000, d=0.46. Extraversion increases by 0.84 from non-project members to project members. There is a significant difference in the Agreeableness when contributors are project members (M=49.1, SD=1.0) compared to when they are non-project members (M=49.3, SD=0.8), paired t(396)=-4.90, p=0.000, d=0.20. Agreeableness decreases by 0.19 from non-project members to project members. There is a significant difference in the Neuroticism when contributors are project members (M=25.2, SD=1.5) compared to when they are non-project members (M=25.6, SD=1.4), paired t(396)=-5.91, p=0.000, d=0.28. Neuroticism decreases by 0.42 from non-project members to project members. Refer Table 21 and Figure 25 for the summary of the observations.

4.2.5 *Discussions*

We started the study by proposing that the personality traits of contributors, mined from software repositories, could offer a way to explain their contributions in different contexts of software development. We were able to present several results that support our idea.

We showed that the personality traits of contributors with different levels of contributions are different. We found a steady increase in the Openness to Experience, the Conscientiousness, and the Extraversion as contribution increases. This is expected as contributors with higher levels of contributions are the ones who take the lead to ensure project success. These contributors characterized by being more insightful, more goal-oriented and social seems intuitive. Further, we found that contributors with higher levels of contributions are lower on the Agreeableness. This is again what we would expect from them to ensure the health of the project.

We found that the project members are more Conscientious, more Extrovert and less Agreeable compared to the non-project members. Again, this is what we would expect from project members that they are rigorous in their work, are social to attract participation, and are less agreeable to maintain quality software development.

We showed that the personality profiles of most active contributors evolve with time. We found that the most active contributors evolve as more Conscientious, more Extrovert and less Agreeable with time. These characteristics explain at the first place why and how these people became the most active contributors.

We showed that the contributors portray different personality traits in different contexts of software development. We found that contributors when working on pull requests are most Conscientious, most Extrovert, most Neurotic and least Agreeable. We believe that this behavior indicates gatekeeping to the entry of quality work and that pull requests mark the highest barrier to entry. Among commits and issues, contributors are most Conscientious, most Neurotic and least Agreeable when working on commits. This implies that after pull requests, commits impose largest barriers to ensure quality software development. It is important to note that contributors are least Extrovert when working on commit. One possible explanation for this is that project members collaborate on commits, unlike issues which are discussed by some prospective project participants. Finally, while working on issues, contributors are least Conscientious, least Neurotic and most Agreeable. This behavior shows how active contributors facilitate the free inflow of suggestions from the masses.

We found that contributors are more Extrovert when they are the member of projects or are reviewing others work. This explains the expectations from the project members and the reviewers of successful projects to maintain cordial terms to attract participation. Other than this, we found that as reporter, contributors are more Open to Experience, more Conscientious, and more Neurotic compared to when they are reviewing others work. Also, we find that when contributors are not the member of the project they more Conscientious, more Agreeable and more Neurotic.

To summarize, our results are promising and encouraging for further explorations. Further improvements required are discussed below. The measures of personality traits used in this study were proposed on texts not specific to software engineering domain. We expect that a software engineering specific lexicon would produce similar effects but with higher effect sizes. However, as rightly pointed out by other studies [185], even an SE-specific lexicon results may not be perfect. Each project is likely to have its own vocabulary and words may have different meanings across different projects [185].

Finally, it is important to note that the observed effect sizes are small. We think that this only illustrates the general problems in finding personality traits from the text. Various highly cited studies have reported effect sizes (Cohen's d) ranging from 0.008 to 0.02 [186] as significant, while our effect sizes range from 0.05 to 0.99. While it is difficult to set a threshold for how big effect size is big enough, we believe that replications of this study will help us achieve a baseline on effect sizes. For instance, personality psychologists routinely calculated and reported effect sizes thereby developing a view of what effect size means and how to interpret it[15].

4.2.6 *Threats to Validity*

4.2.6.1 *Internal Validity*

We assume a causal relationship between the personality of contributors and their written communications to discuss issues, pull requests and commits on GitHub. This assumption is grounded on the empirical evidence in other domains [175]. Also, this assumption has been widely validated through its applications on various platforms like Facebook [178], Twitter [164] [179], StackOverflow [163] [180], etc.

15 https://funderstorms.wordpress.com/2013/02/01/does-effect-size-matter/

The genuineness of the personality traits mined is prone to adversary effects where people try to present themselves in ways that best suits their purpose. However, we believe that this is not a concern for this study as the contributors are not aware that their communications are being monitored. Further, we collected data for an extended period of time to strengthen the validity of our results.

This is an empirical study and we do not claim any causality between the inferred personality traits and the various factors analyzed in this study. However, the fact that the findings match our intuitions strengthens our believe in the results obtained.

4.2.6.2 *Construct Validity*

Mining personality traits from discussions on the issues, pull requests and commits are challenging. We adopted approaches used in existing studies to measure the personality traits from communications. While these approaches are validated on various studies, the measures of personality traits may be biased against contributors whose first language is not English.

We find that a subset of contributors never participated in any discussions. The personalities of these contributors cannot be analyzed using the techniques proposed here and are excluded from analyses. Any traits specific to these contributors are not captured and hence not presented here.

4.2.6.3 *Reliability Validity*

This is the first attempt to study the relationship of the personality traits with the contribution and the context of software development. There exists no ground truth with which we can compare our findings. However, the match between our findings and our intuitions strengthens our confidence in the reliability of the results. Further, while the measures of personality traits from written communications were not designed for software engineering domain, we believe that its applications in various domains increases our confidence on the results obtained.

4.2.6.4 *External Validity*

We analyzed the most actively discussed projects and the most active contributors. These projects and contributors are not representative of all the open source projects. So, while the results of our study may not generalize, the framework presented here can be applied to different contexts. Replication of this work on a wider range of open source projects and commercial projects will confirm our findings.

4.3 ROLE AND PAST CONTRIBUTIONS

4.3.1 *Introduction*

Contributors leaving project incurs significant direct and indirect costs [187] [188][54][57] thereby making it critical to retain existing contributors [189]. A data-driven approach to study future participation helps overcome challenges in existing practices by providing objectivity and transparency. In this work, we examine contributor characteristics namely role of participation and amount of work done as a measure of predicting future participation. We present an approach that uses statistical measures to classify contributors based on contribution into three mutually exclusive sets namely non-core team, loose core team and tight core team. Also we define attrition as a function of participation in two consecutive time intervals and study attrition rate for four roles (reporter, owner, commenter and cc'ed-contributor (cc'ed)) and three classes of contributors. We conduct experiments on Google Chromium Issue Tracking System (GC-ITS) dataset[16]. The dataset is extracted for four consecutive years and observations are recorded quarterly (3 months). In Issue Tracking Systems contributors play various roles. However, for this work we focus on four roles namely reporter, owner, commenter and cc'ed. These roles are associated with various stages of bug fixing lifecycle. Reporter reports the issue. Issue is fixed by owner in collaboration with commenters participating via threaded discussion forum. Owner may also request participation by cc'ing contributors. Contributors cc'ed are called for to serve specific request in issue.

16 https://code.google.com/p/chromium

Table 23: Role based individual contribution pattern

Role	Mean	Std	Min	Max	.25Q	.5Q	.75Q	.9Q	.95Q	.99Q	Skew
Own	24.8	038.1	1	000559	3	10	32	67	94	172.1	003.8
Rep	02.9	008.8	1	000476	1	01	02	04	11	040.0	013.5
CC	20.0	051.4	1	001315	1	04	20	59	92	190.2	009.7
Com	14.0	422.9	1	119803	1	01	02	07	30	239.0	129.9

4.3.2 *Empirical Analysis*

CLASS OF CONTRIBUTION Research shows that open source projects follow Pareto Distribution [53] that is 20% of contributors do 80% of work. However, our experimental results do not communicate the same. Table 23 shows that in GC-ITS contribution pattern is highly skewed for four roles. This observation demands statistical and data-driven approach to classify contribution of contributors. In Algorithm 1 we classify each contributor in one of the three mutually exclusive classes namely non-core team, loose core team and tight core team based on contribution. Non-Core Team (NCT) includes contributors who join project to address some specific issue they encountered. Loose Core Team (LCT) includes dedicated contributors with substantial contribution and Tight Core Team (TCT) includes contributors with relatively large contribution (with respect to LCT).

Algorithm 1 Algorithm to Identify Contribution Class

```
Require: struct{Owner O, Reporter R, CC'ed CC, Commenter Com} Contributor Con[]
Ensure: ConClass[][3]
 1: procedure ContributionClass(Con)
 2:     for all Time-Interval $T_t$ do
 3:         Normalize contribution using Range Normalization [1-100]
                $\boxed{Y_i = \frac{(100-1) \times X_i}{Max(X) - Min(X)}}$        (1)
 4:         Calculate weighted Geometric Mean where $w_O > w_R > w_{CC} >= w_{Com}$ and sum($w_O + w_R + w_{CC} + w_{Com}$) =1
                $\boxed{Score(S) = O^{w_O} \times R^{w_R} \times CC^{w_{CC}} \times Com^{w_{Com}}}$        (2)
 5:         Calculate Z-Score
                $\boxed{Z = \frac{S - \mu}{\sigma}}$        (3)
 6:         if $Z < 0$ then
 7:             $ConClass_{T_t}$[1] ← Con[Z< 0]        ▷ Non-Core Team
 8:         else if $Z > 1$ then
 9:             $ConClass_{T_t}$[2] ← Con[Z > 1]        ▷ Tight Core Team
10:         else
11:             $ConClass_{T_t}$[3] ← Con[Z >= 0 && Z <= 1]        ▷ Loose Core Team
12:         end if
13:         ConClass[$T_t$] ← $ConClass_{T_t}$
14:     end for
15:     return ConClass
16: end procedure
```

The input to the Algorithm 1 is contribution of contributors where each contributor plays at least one of the four roles. The output is contribution class of contributors calculated for all time-intervals. We measure the contribution for the role of owner, reporter and cc'ed as the total number of issues participated in time-interval defined quarterly. Similarly, for commenter we measure contribution in terms of total number of comments in time-interval. Further to ensure homogeneity for cross comparison we range normalize contribution in each role for all time-intervals on a scale of 1-100 (refer Equation 1). We then append the scores for four roles of contributor for a time-interval to create a structure. All missing values are assigned a negligibly small value (0.0001). We generate cumulative score that measures contribution in terms of relevance of role (weight of owner (W_O) > weight of reporter (W_R) > weight of cc'ed (W_{CC}) > weight of commenter (W_{Cm})) as W_O=0.5, W_R=0.25, W_{CC}=0.125 and W_{Cm}=0.125. The choice of weight depends on specific requirements and may vary for individuals. Assuming that participation in one role is independent of participation in other roles, we use weighted Geometric Mean to generate cumulative score (refer to Equation 2).The score generated ranges from 100 (highest) to approximately 0.0001 (negligible or no contribution).

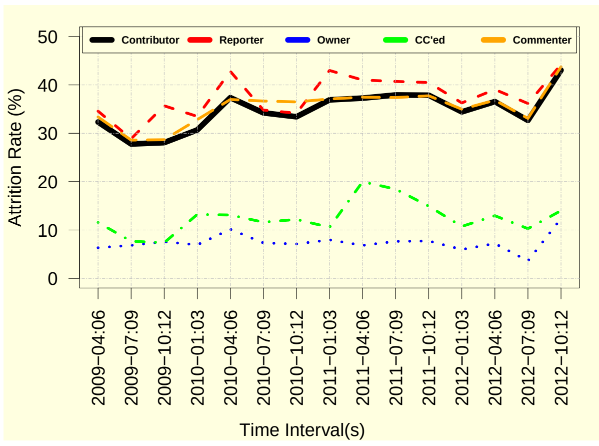


Figure 26: Attrition rate of maintainers for four years

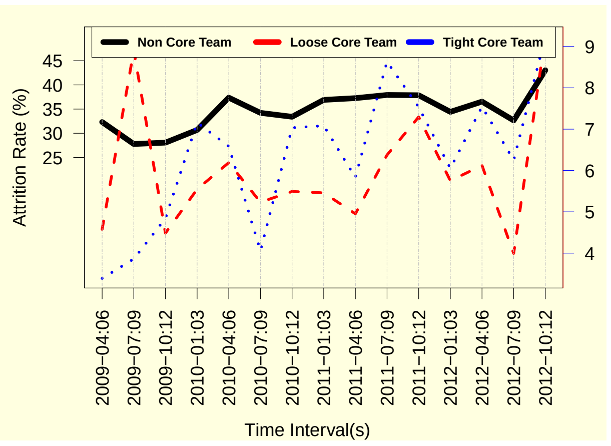


Figure 27: Comparison of attrition rates for contribution classes for four years

Next we find relative relevance of contribution that is the number of standard deviations datum is related to mean. We calculate Z-Score (refer to Equation 3). If value of Z is less than 0, it indicates that contributor is part of NCT. If value of Z is greater than 1 it defines TCT. Likewise value of Z greater than equal to 0 and less than equal to 1 implies LCT. We calculate contribution class for all time-intervals and return set of contributors for each class for each time-interval.

ATTRITION RATE In this study, we believe that the contributor has left the project if duration of inactivity exceeds one time-interval (in this case measured quarterly). Thus Attrition Rate (AR) for time-interval T_t measures (in percentage) the fraction of contributors who left the project in time-interval T_t to the total number of contributors who participated in time-interval T_t and its preceding time-interval T_{t-1} .

GRAPHICAL ANALYSIS OF THE RELATIONSHIP BETWEEN CONTRIBUTOR CHARACTERISTICS AND FUTURE PARTICIPATION In Figure 35 horizontal axis of the plot represents consecutive time-intervals (measured quarterly) and vertical axis shows attrition rate. Colored lines (refer to legend) present attrition rate for contributors and their roles. We observe that contributor attrition rate (irrespective of roles as shown in black) fluctuates from 27% to 47%. Also we observe marked difference in attrition patterns for four roles. We see minimum attrition rate for owner (shown in blue) and maximum for reporter (shown in red). This follows the intuition that not every contributor can own issues. Figure 27 compares attrition rate of three classes of contributors namely non-core team, loose core team and tight core team (refer Algorithm 1) across four years. We see in Figure 27 that the attrition rate for LCT and TCT ranges from 3% to 10% which is relatively less than the attrition rate for NCT (ranges between 27% and 43%). It indicates that retention in project is directly related to degree of involvement in project. Also interestingly after initial fluctuations, attrition rate of TCT is higher than attrition rate of LCT indicating that TCT contributes relatively large however sporadically.

4.4 SUMMARY

In this chapter we studied the influence of competing projects' dynamics on contributor participation, effect of personality traits on levels of contributions, and impact of role reputation and contribution on developer participation.

First, we studied the influence of diverse project and contributor characteristics on future participation. A large-scale study of 2,217 projects hosted on GitHub shows that 1 in every 5 original projects observes a decline in the sustainability of the developer community participation after forking. We find that the negative effect is more pronounced in projects ported to GitHub from other platforms (≈ 20%), compared to GitHub developed projects (≈ 9%). We also find that the observed behavior can be explained in terms of the characteristics of the competing projects at the time of forking. For instance, in medium sized projects an increase in the maturity of the original project by a year decreases the odds of decline in the sustainability of the developer participation by 23%.

Second, we analyzed the relationship of personality traits of contributors with their levels of contributions. We found that personality traits of contributors relate to their contributions and that contributors behave differently in different work situations. We analyzed 243 most actively discussed projects and 423 most active contributors and showed that:

- Contributors with different levels of contributions have different personality profiles.
- Personality profiles of most active contributors evolve with time.
- Contributors portray different personality traits in different contexts of software development.

The findings of our study are promising. Most of the observations confirm our intuitions, thereby demonstrating the inferential power of personality traits in explaining the behavior in various contexts of software development.

Finally, we studied the impact of role reputation and contribution on developer participation. We measured contributor characteristics and investigated their relationship with contributor attrition by mining Issue Tracking System. Experimental results show that the likelihood of future participation increases with increase in relevance of role in project and level of participation in previous time-interval.

5 MEASURES OF SOFTWARE CONTRIBUTOR PRODUCTIVITY

Evaluating software contributor productivity is a standard practice in organizations. It is used to understand the value addition by various contributors and its influence on project success. However, accurate measurement of contributions based on pre-defined objectives, roles and key performance indicators is a challenging task. Existing approaches to measure contributor productivity are subjective and imprecise. There is a need to objectively measure software contributor productivity. In this chapter, we identify 1) gaps between expectations and measurements of contributions in practice, 2) propose metrics to measure individual contribution and team participation, 3) present visualizations to help scrutinize underpinning factors involved in explaining productivity, 4) evaluate the proposed metrics and 5) discuss its applications. Here, we study individual and team productivity in the software maintenance team of the Google Chromium project.

5.1 INTRODUCTION

"What you cannot measure you cannot control" sets the broad motivation of the work presented in this study. Assessing productivity of employees and workers is a standard human resource management practice followed in organizations world-wide [190]. It is an important and routine activity performed within organizations to measure value addition based on pre-defined Key Performance Indicators (KPIs) [191] [192]. These measures are required for career advancement decisions, identification of strengths and areas of improvements of the employee, rewards and recognition, resource planning etc. [191] [192].

Assessing productivity of people is fraught with several challenges and imperfections that negatively influence employees' motivation and organization's growth [190][193]. Solutions for contribution and performance assessment of software maintenance professionals as well as defect-fixing performance is an area that has recently attracted several researcher's attention [194] [36] [34] [35] [37] [67] [26][33]. This study is motivated by the need to develop novel framework and metrics to accurately and objectively measure contribution of software maintenance professionals by mining data archived in Issue Tracking Systems. The aim of this study are the following:

1. To investigate the perceived importance of key performance indicators and their measurements in practice.
2. To investigate role-based metrics to accurately, reliably and objectively measure the productivity of software maintenance professionals involved in defect fixing process.
3. To validate the proposed metrics with software maintenance professionals in industry and apply it on real-world dataset demonstrating the effectiveness of the approach.
4. To visualize the contributions of software maintenance professionals in the contexts of software development.
5. To investigate the effectiveness of the proposed visualization solution by studying the complex interplay of variables to 1) analyze trends, outlier behavior, patterns and regularities, 2) extract actionable insights from the perspectives of decision makers and 3) get their validation.
6. To investigate metrics to objectively characterize community stability on key stability indicators by mining Issue Tracking System.
7. To demonstrate the inferential ability of time series data on key stability indicators by investigating the stability of the community and estimating future participation.

5.2 RELATED WORK AND RESEARCH CONTRIBUTIONS

The literature on evaluating contributor productivity can be broadly classified into three heads: 1) measure individual productivity, 2) measure team productivity and 3) approaches to visualize individual and team productivity.

5.2.1 *Measure Individual Productivity*

The most closely related research to the study presented here is the work by Nagwani et al. However, there are several differences between the work by Nagwani et al. and this study. Nagwani et al. propose a team-member ranking technique for software defect archives [37]. Their technique consists of generating a ranked list of developers based on individual contributions such as the quantity and quality of bugs fixed, comment activity in discussion forums and bug reporting productivity [37].

Gousios et al. present an approach for measuring developer's involvement and activity by mining software repositories such as source code repository, document archives, mailing lists, discussion forums, defect tracking system, Wiki and IRC [36]. They present a list of pre-defined developer actions (easily measurable and weight assigned based on importance) such as reporting a bug, starting a new wiki page, committing code to the source-code repository which are used as variables in a contribution factor function [36].

Ahsan et al. present an application of mining developer effort data from bug-fix activity repository to build effort estimation models for the purpose of planning (such as scheduling a project release date) and cost estimation [194]. Their approach consists of measuring the time elapsed between assigned and resolved status of a bug report and multiplying the time with weights (incorporating defect severity-level information) to compute developer contribution measure [194].

Kidane et al. propose two productivity indices (for online communities of developers and users of the open source projects such as Eclipse) in their study on correlating temporal communication patterns with performance and creativity: creativity index and performance index [35]. They define creativity index as the ratio of the number of enhancement (change in the quantity of software) integrated in a pre-defined time-period and the number of bugs resolved in the same time-period [35]. Performance index is defined as number of bugs resolved in a pre-defined time-period and number of bugs reported in the same time-period.

Kanij et al. mention that there are no well established and widely adopted performance metrics for software testers and motivate the need for a multi-faceted evaluation method for software testers [34]. They conduct a survey of industry professionals and list five factors which are important for measuring performance of software testers: number of bugs found, severity of bugs, quality of bug report, ability of bug advocacy and rigorousness of test planning and execution [34].

Rigby et al. present an approach to assess the personality of developers by mining mailing list archives [67]. They conduct experiments (using a psychometrically-based word count text analysis tool called as Linguistic Inquiry and Word Count) on Apache HTTPD server's mailing list to assess personality traits of Apache developers such as diligence, attitude, agreeableness, openness, neuroticism and extroversion [67].

Fernandes et al. mention that performance evaluation of teams is challenging in software development projects [33]. They propose an analytical model (Stochastic Automata Networks model) and conduct a practical case-study in the context of a distributed software development setting [33].

5.2.2 *Measure Team Productivity*

FLOSS projects undergo phases of contributor evolution. In initial phase, a small core team develops software. As the project matures multiple core teams contribute to develop software [195][52]. Research shows that a majority of FLOSS projects fails due to the lack of sustained developer participation [63][57].

To estimate the stability of FLOSS projects, research identifies three community stability indicators namely attrition, retention, and regeneration [53] [63][57][56][189]. While the three indicators are known to explain the stability, there does not exists metrics to quantify contributor churn.

In another line of study, researches compute the likelihood of future participation by mining software repositories[12][63][11][59]. Studies by Wu et al. and Yu et al. conduct time series analysis of contributor participation to identify trends and use it for predicting future participation[196][197]. However, existing studies do not explain how changes in community participation pattern may affect stability of community.

5.2.3 *Visualize Contributor Productivity*

Visualization ease software understandability and analyzes of the evolution of software ecosystem [43][198]. It also find its application in analyzing software team. To facilitate visualization in management domain Zhang et al. proposed a theoretical, general visualization model. They identify five processes namely domain problem space analysis, domain data and knowledge collection, pattern discovery and data aggregation, and image construction as theoretical foundations to achieve data comprehension and improve problem solving performance [199]. Schonhage et al. build a prototype for visualization in business to aid managers analyze past and present data and venture with future situations [200].

Storey et al. proposed a framework to describe, compare and understand human centric awareness visualization tool in software development [26]. Taylor et al. introduced the concept of author entropy to characterize authorship. Author entropy in conjunction with other software metrics has the potential to identify areas of concerns within source code [38]. Gilbert et al. explored group dynamics to compare developers and their contribution in a distributed software community by analyzing social visualization code [9]. Robles et al. propose a novel methodology to visually analyze evolution of core team to ensure smooth transitions by identifying breakpoints and unevenness [54]. However, to the best of our knowledge existing literature does not explore the expressive power of visualization to analyze productivity by mining software repositories.

In context of literature, the work presented here makes the following novel contributions:

1. A framework for assessing the productivity of software maintenance professionals. We propose 11 metrics for four different roles. The novel contribution of this work is to assess the developer in terms of the various roles played by a developer. The metrics (data-driven and evidence-based) is computed by mining data in an Issue Tracking System. While there are some studies on the topic of contribution assessment for Software Maintenance Professionals, we believe (based on our literature survey) that the area is relatively unexplored and in this study we present a fresh perspective to the problem.

2. A survey conducted with experienced software maintenance professionals on the topic of assessing contributor productivity for bug reporter, bug owner, bug triager and contributor. We believe that there is a dearth of academic studies surveying the current practices and metrics used in the industry for contribution assessment of software maintenance professionals. To the best of our knowledge, the work presented in this study is the first study to present the framework, survey from practitioners and application of the proposed framework on a real-world publicly available dataset from a popular open-source project.

3. We propose 6 visualization techniques for the four roles of software maintenance professionals viz. bug reporter, bug triager, bug owner and bug collaborator. The unique contribution here is a visualization framework (panoramic view) to justifiably distinguish performance by deriving actionable insights. We see application of visualization in software industry to analyze process and generate awareness. However, its application to measure productivity of software maintenance professionals by mining software repositories is unexplored to the best of our knowledge.

4. Implementation of proposed visualization techniques on real world data of GC-ITS, infer actionable insights and validate its usability based on survey responses from professionals.

5. A framework to quantify key stability indicators on community participation patterns; investigate trends and predict stability of software maintenance projects as identified by mining Issue Tracking System.

5.3 METHODOLOGY

We believe that inputs from practitioners are needed to inform our research. So we conduct a survey of two experienced industry professionals (project manager level with more than a decade of experience in software development and maintenance) and present the results of the survey as well as our insights (refer to Section 5.4.1). We present 11 performance and contribution metrics for four roles: bug reporter, bug triager, bug owner and collaborator. We define each metrics and describe the rationale and the formula (Section 5.5). The metrics are discussed at an abstract level and are not specific

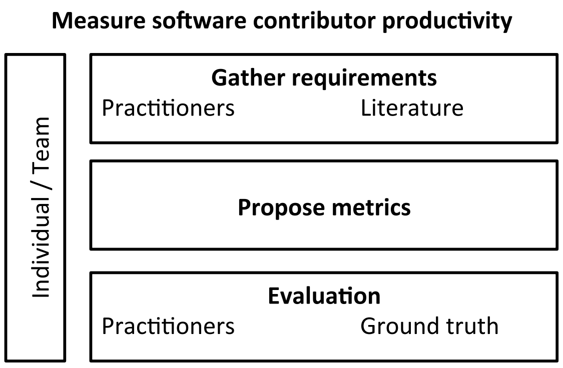


Figure 28: Research method used to measure software contributor productivity

to a project or Issue Tracker. Next, we use the general visualization model proposed by Zhang et al. [199] to propose a framework of visualizations. We start by studying variegated sources for unconventional visualizations. We identify approximately 30 unique visualizations. For each visualization, we analyze the rationale behind its proposition and the nature of information that can be inferred from the visualization. We map these visualizations to the metrics to capture multiple information needs of decision makers. Each visualization technique maps to multiple metrics. However, due to limited space availability and for the ease of comprehension we present one metric for a visualization (refer Section 5.4.6). All visualizations present in this work are written in R language[1] (GNU project for statistical computing and graphics). Post-implementation, we conduct survey of practitioners. We received 10 valid responses from practitioners in large global IT industry. 9 survey respondents have more than 5 years of experience and 1 had more than 1 year of experience. 8 survey respondents played role of Project Manager, 1 was head of R & D initiatives and 2 Bug Owner/Bug Fixer with overlapping roles. 8 out of 10 survey respondents had been appraisers in the past or are in present.

We also propose metrics to analyze temporal community contribution. We identify three KSIs namely attrition, regeneration, and retention from the literature. We propose metrics to quantify the three KSIs by mining ITS. A contributor can play multiple roles by participating in multiple activities. For instance, a contributor may report an issue (reporter), and also own it (owner). A contributor may comment on an issue (commenter) or may collaborate because contributor is cc'ed the issue (collaborator-cc). In this study, we investigate participation in four roles, and present the results. The list of four roles is representative and not exhaustive. Also the roles are overlapping and by no means represent disjoint sets. We generate time series data of metrics and investigate the inferential ability of the temporal trends. We present representative research questions to investigate the inferential ability of the metrics. We model community participation pattern to estimate future participation by using statistical prediction models with justifications for the choice of models (refer Section 5.5.4). We assess the goodness of fit of the models, compute the accuracy of prediction, compare models, and present our inferences.

In short, we gather requirements for the study by talking to practitioners or from literature. We propose metrics that meet the requirements identified in stage one. Then we evaluate it based on inputs from practitioners or by examining the characteristics of project. Table 28 summarizes the research method used to measure individual and team productivity.

Experimental dataset

The dataset for analyzing the results is build on GC-ITS. The choice of dataset is motivated by the following reasons: 1) publicly available, 2) dataset is replicable (can be downloaded using Issue Tracking System API), 3) wide usage and popularity. We implement the proposed metrics and visualization on the Google Chromium dataset. Unless specified otherwise all analysis is conducted on 'Closed' bug reports with status 'Fixed' or 'Verified'.

To evaluate the productivity of software maintenance professionals, we analyze one year data. This duration is in agreement with the performance appraisal cycle at most of the places. The details of the dataset are given in Table 24. The one year dataset consists of 24,743 bugs, 6,765 reporters, 500

1 http://www.r-project.org

Table 24: Experimental Dataset 1

Duration		1year	
Start date of the first bug report in the dataset	01/01/2009	End date of the last bug report in the dataset	31/12/2009
Bug ID of the first bug report	5,954	Bug ID of the last bug report	31,403
Total number of Reported bugs		25,383	
Total number of Forbidden bugs		640	
Total number of Available bugs		(25,383-640)=24,743	

Table 25: Experimental Dataset 2

Duration		1year	
Start date of year-I	01/01/2009	End date of year-I	31/12/2009
Start date of year-II	01/01/2010	End date of year-II	31/12/2010
Year 2009-10		Year 2010-11	
Available bugs	24,629	Available bugs	36,176
CLOSED bugs	21,984	CLOSED bugs	32,289
Statistics for CLOSED bug reports (Count of)			
Bug Reporters	6,742	Bug Reporters	13,581
Bug Triagers	325	Bug Triagers	475
Bug Owners	307	Bug Owners	480
Bug Collaborators	12,324	Bug Collaborators	15,307

triagers, 391 owners, and 5,043 collaborators. Out of the 24,743 reported bugs, 8,628 bug reports were Closed with status Fixed or Verified.

Next, to help visualize the trend, we analyze contribution in two consecutive years (2009-11) with 34,056 unique contributors. However dataset to create visualization for a role encompass less number of developers compared to total (refer Table 25). It happens due to 1) exclusion of data points with missing or inconsistent information 2) considering developers who worked for two years and 3) setting thresholding criteria on selection of developers for analysis.

Finally, to analyze temporal contributor community participation, we conduct experiments on four years data of Google Chromium Issue Tracking System (GC-ITS) extracted from January 1, 2009 to December 31, 2013 and measure it quarterly. In Google Chromium project, we observe that contributor participation is intermittent in time. A contributor may continue participation months after discontinuing participation in the project. We assume that contributor has left the project when contributor does not participate for at least three months. This may generate false positives as contributors continue participation quarters after last contributor. Moreover, this behavior is exhibited by the majority of participants (70% of contributors in FLOSS projects are one-time contributors or contribute sporadically in time [54]). Thus, assuming lag in participation as contributor churn we portray community behavior.

5.4 INDIVIDUAL KEY PERFORMANCE INDICATORS - MEASUREMENT AND VISUALIZATION

In this section, we describe 11 performance and contribution assessment metrics for four different roles. These 11 metrics are representative metrics valued by the organization and are not exhaustive.

Table 26: Experimental Dataset 3

Start Date	End Date	Available Bug Reports	Contributor Count
2009-01-01	2009-12-31	025841	13006
2010-01-01	2010-12-31	044041	26654
2011-01-01	2011-12-31	051683	28170
2012-01-01	2012-12-31	057970	30510

Table 27: Defining the metadata of the survey

Attribute	Definition	Scale
PIK	Perceived Importance of Key Performance Indicators	5: Highly influence (important)
		1: Does not influence (least important)
CopKC	Current organizational practices to capture KPIs	1: Captured accurately (formula based i.e. objective measure)
		2: Captured accurately (based on subjective information, self-reporting and perception)
		3: Unable to capture accurately
		4: Not relevant / Not considered

Following are few introductory and relevant concepts which are common to several metrics definition and Google Chromium project describes these in bug-lifecycle and reporting guidelines[2].

A closed bug can have status values as: Fixed, Verified, Duplicate, Won'tFix, ExternalDependency, FixUnreleased, Invalid and Others[3]. An issue can be assigned a priority value from 0 to 3 (0 is most urgent; 3 is least urgent). Google Chromium issue reports has a field called as Area[4]. Area represents the product area to which an issue belongs. An issue can belong to multiple areas such as BuildTools, ChromeFrame, Compat-System, Compat-Web, Internals, Feature and WebKit. A product area BuildTools maps to Gyp, gclient, gcl, buildbots, trybots, etc. and the product area WebKit maps to HTML, CSS, Javascript, and HTML5 features.

5.4.1 *Survey of Software Maintenance Professionals*

We conducted a survey to understand the relative importance of various Key Performance Indicators (KPIs) for the four roles: bug reporter, bug owner, bug triager and collaborator. We prepared a questionnaire consisting of four parts (refer to Tables 28, 29, 30, 31). Each part corresponds to one of the four roles. The questionnaire for each part consists of a question and two response fields. The question mentions a specific activity for a particular role. For example, for the role of bug owner the activity is: number of high priority bugs fixed, and for the role of bug triager the activity is: number of times the triager is able to correctly assign the bug report to the expert (bug fixer who is the best person to resolve the given bug report).

One of the response fields, Perceived Importance of Key Performance Indicators (PIK) denotes the importance of the activity (for the given role) on a scale of 1 to 5 (5 being highest and 1 being lowest). The other response field, Current Organizational Practices to capture KPIs (CopKC) denotes the extent to which these performance indicators are captured in practice. We broadly define the notion of the ability to capture KPIs into two heads: non-relevant and relevant. The KPI is either considered non-relevant (for evaluation of developers) and hence not measured or it is considered important. The relevant KPIs are either not captured (inability of the organizations to measure) or they are able to capture it. The KPIs captured can again be classified based on the approach to measure it as objective or subjective. Objective measures are captured using statistics (based on formulas) while subjective measures are evaluated based on the perceived notion of contribution. This hierarchical classification completely defines the notion of measurement in organization (refer to Table 27 specifying the metadata for the survey).

We surveyed two experienced software maintenance professionals for this study. The two professionals had more than 10 years of experience in the software industry and were at the project manager level. One of them worked in a large and global IT (Information Technology) services company (more than 100 thousand employees) and the other worked in a small (about 100 employees) offshore product development services company. Tables 28, 29, 30 and 31 show the results of the survey. Some of the cells in the table have values '−' or 'NULL'. '−' means that at the time of the survey, the survey respondent was not asked this question and 'NULL' implies that the survey respondent did not answer this question (left the field blank).

We find gaps between the perceived importance of certain performance indicators and the extent to which such indicators are objectively and accurately measured in practice. We believe that the perfor-

2 http://www.chromium.org/for-testers/bug-reporting-guidelines
3 user defined Status as permitted by Google Chromium Bug Reporting Guidelines
4 http://www.chromium.org/for-testers/bug-reporting-guidelines/chromium-bug-labels

Table 28: Survey results for the role of bug reporter

Survey questions	Company A		Company B	
	PIK	CopKC	PIK	CopKC
Number of bugs reported	4	1	3	1
Number of Duplicate bugs reported	5	-	4	-
Number of non-Duplicate bugs reported	3	-	5	-
Number of Invalid bugs reported	3	-	5	-
Number of high Priority bugs reported	5	NULL	4	1
Number of high Severity bugs reported	5	NULL	5	1
Number of bugs reported that are later reopened	NULL	NULL	4	1
Number of hours worked	-	3	-	-
Quality of reported bugs	5	3	5	2
Followed bug reporting guidelines	5	2	5	2
Correctly assigned bug area (like WebKit, BrowserUI etc.)	5	NULL	4	3
Correctly assigned bug type (like Regression, Performance etc.)	4	NULL	5	3
Reported bugs across multiple components (Diversity of Experience)	4	3	2	3
Reported bugs belonging to a specific component (Specialization)	NULL	3	4	3
Participation level delivered (responded to queries and clarifications) after reporting bug	-	3	-	3

Table 29: Survey results for the role of bug owner

Survey questions	Company A		Company B	
	PIK	CopKC	PIK	CopKC
Number of bugs assigned or owned	5	-	4	-
Number of bugs successfully resolved (from the set of bugs owned)	5	NULL	5	1
Number of high Priority bugs owned	5	1	4	3
Number of high Severity bugs owned	5	3	4	1
Number of resolved bugs that get reopened	5	1	4	1
Number of hours worked	-	1	-	3
Participated in (facilitated) discussion through comments on Issue Tracker	4	-	3	-
Owned bugs across multiple components (Diversity of Experience)	5	3	3	3
Owned bugs belonging to a specific component (Specialization)	5	3	3	3
Average time taken to resolve bug	5	NULL	4	2
Response time to a directly addressed comment	-	3	-	NULL

Table 30: Survey results for the role of bug collaborator

Survey questions	Company A		Company B	
	PIK	CopKC	PIK	CopKC
Participation level delivered (responded to queries and clarifications) through online threaded discussions	3	NULL	4	1
Response time to a directly addressed comment on Issue Tracker	4	NULL	5	3
Collaborated in bugs across multiple components (Diversity of Experience)	2	4	2	3
Collaborated in bugs belonging to a specific component (Specialization)	3	2	2	3
Average time taken to resolve bug	5	3	1	2
Number of times collaborated on high Priority bugs	5	NULL	3	2
Number of times collaborated on high Severity bugs	5	NULL	3	2
Number of hours worked	-	3	-	3

Table 31: Survey results for the role of bug triager

Survey questions	Company A		Company B	
	PIK	CopKC	PIK	CopKC
Number of bugs Triaged	5	NULL	5	2
Number of hours worked	-	2	-	3
Identified Duplicate bug reports	5	NULL	4	2
Identified Invalid bug reports	5	NULL	4	2
Identified exact bug area (like WebKit, BrowserUI etc.)	5	3	5	3
Assigned best developer considering skills and workload	5	-	5	-
Correctly assigned owner (fixer)	-	3	-	3
Correctly assigned bug type (like Regression, Performance etc.)	4	2	5	2
Correctly assigned Priority/Severity	5	3	5	3
Participated in (facilitated) discussion through comments on Issue Tracker	4	-	4	-
Response time to a directly addressed comment on Issue tracker	-	3	-	3

mance indicators considered important are not measured rigorously due to the lack of tool support. Results of the survey (refer to Tables 28, 29, 30 and 31) are evidences supporting the need for developing contribution and performance assessment framework for Software Maintenance Professionals.

It is interesting to note that the two experienced survey respondents from different sized organizations have given similar values to some KPIs while for the others, the result varied considerably. Based on the survey results, we infer that the perceived value of an attribute varies across organizations. Therefore, the proposed metrics should account for the perceived value of the attribute for the organization while measuring the contribution of a developer for a given role.

5.4.2 *Bug Owner Metrics*

PRIORITY WEIGHTED FIXED ISSUES (PWFI) We propose a contribution metric for the bug owner which incorporates the number of bugs fixed as well as the priority of each bug fixed. For each bug owner in the dataset, we count only Fixed or Verified issues and not Duplicate, Won'tFix, ExternalDependency, FixUnreleased and Invalid. We exclude Invalid and Won'tFix bugs because these bugs are not a contribution by the bug owner to the project.

$$\boxed{PWFI(d) = \sum_{i=0}^{|P|} W_{P_i} \times \frac{N^d_{P_i}}{N_{P_i}}} \tag{4}$$

where W_{P_i} is the weight (a tuning parameter which is an indicator of importance, higher for urgent bugs and lowest for least urgent bugs) or multiplication factor incorporating issue priority information. Due to the weight W_{P_i} (sum of the weight equal to one), a bug-owner contribution is not just a function of absolute number of bugs fixed and is also dependent on the type of bugs fixed. N_{P_i} is the total number of Fixed or Verified bugs in the dataset with priority P_i and $N^d_{P_i}$ is the number of priority P_i bugs Fixed or Verified by developer d in the dataset. The value of $PWFI(d)$ will be between 0 and 1 (higher the value, more is the contribution).

SPECIALIZATION AND BREADTH INDEX (SBI) A developer can specialize and be an expert in a specific product area (component) or can have knowledge across several product areas. Let n be the total number of product areas and d denotes a developer. We represent $B_n(d)$ as an index for breadth of expertise and $1 - B_n(d)$ as an index for specialization. p_i is the probability value (probability of developer d working on component i) derived from historical dataset. A lower value of $B_n(d)$ means specialization and a higher value means breadth of expertise of a developer.

$$\boxed{B_n(d) = -\sum_{i=1}^{i=n} p_i * \log_n(p_i)} \tag{5}$$

Table 32: Priority-Time correlation statistics

Priority	Count	Time Duration (in days)				
	(of bugs)	Maximum	3/4 Quartile	Median	1/4 Quartile	Minimum
P0	00258	0495.37	010.69	02.71	0.44	0.0002
P1 ↑	02,839	1120.37	044.10 ↓	11.04 ↓	2.03	0.0000
P2	17,923	1273.15	129.63	21.99	1.93	0.0000
P3	02,265	1180.56	206.02	55.90	5.03	0.0000

The $B_n(d)$ value can vary from 0 (minimal) to 1 (maximum). If the probability of all areas is the same (same distribution) then the $B_n(d)$ is maximal. This is because the p_i value is the same for all n. On the other extreme end, if there is only one area associated to a particular developer d then the $B_n(d)$ becomes minimal (value of 0). The interpretation is that when $B_n(d)$ is low for a specific developer then it means a small set of product areas are associated with the specific developer d.

DEVIATION FROM MEDIAN TIME TO REPAIR (DMTTR) Bugs reported in Issue Tracking System can be categorized into four classes as adaptive, perfective, corrective and preventive[5] (as part of Software Maintenance Activities). Depending on their class these bugs have different urgency with which they must be resolved. For instance, a loophole reported in security of a software must be fixed prior to improving its Graphical User Interface (GUI).

According to Google Chromium bug label guidelines, priority of a bug is proportional to the urgency of the task. More urgent a task; higher is its priority. Also, urgency is a measure of the time required to fix bug. Therefore, high priority bugs should take less time to repair as compared to low priority bugs (more urgent tasks take less time to fix). To support this assertion, we conducted an experiment on Google Chromium Issue Tracking System dataset to calculate the median time required to repair bugs with same priority. The results of the experiment support the assertion (as shown in Table 32).

One of the key responsibilities for the role of bug owner is to fix bugs in time where time required may be influenced by various external factors. Bettenburg et al. in their work pointed out that a poor quality reported bug increases the efforts and hence the time required to fix it [201] [202]. Irrespective of these factors, ensuring timely completion of the reported bugs is an indicator of positive contribution for the role of bug owner and vice-versa.

DMTTR metric is defined for Closed bugs with status Fixed or Verified. It calculates deviation of the time required to fix a bug from the median time required to repair bugs with same priority.

$$\boxed{\begin{aligned} &DMTTR(o,p) = \frac{1}{T_{total}} \times \sum_{i=0}^{|P|} \sum_{\forall b_i} w_i \times (T_{b_i} - MTTR_{P_i}) \\ &where \begin{cases} |T_{b_i} - MTTR_{P_i}| & \text{if } (T_{b_i} - MTTR_{P_i}) < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}} \tag{6}$$

$$\boxed{\begin{aligned} &DMTTR(o,n) = \frac{1}{T_{total}} \times \sum_{i=0}^{|P|} \sum_{\forall b_i} w_i \times (T_{b_i} - MTTR_{P_i}) \\ &where \begin{cases} |T_{b_i} - MTTR_{P_i}| & \text{if } (T_{b_i} - MTTR_{P_i}) > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}} \tag{7}$$

Time required to fix a bug may be less than, greater than or equal to the median time required to fix same priority bugs. Less or equal time required to fix a bug is a measure of positive contribution and vice-versa. We propose two variants of DMTTR(o) i.e. DMTTR(o,p) and DMTTR(o,n). For an owner o, DMTTR(o,p) measures positive contribution p (relatively less time to repair) and DMTTR(o,n) measures negative contribution n (relatively more time to repair). w_i is a normalized tuning parameter which weighs deviation proportional to priority i.e. less time required to fix a high priority bug adds

5 http://en.wikipedia.org/wiki/Software_maintenance

more value (to the contribution of a bug owner) than less time required to fix a low priority bug. Here b_i is a bug report with priority P_i. T_{b_i} is the total time required to fix bug b_i and is measured as difference in the reported timestamp of bug as Fixed and first reported comment with status Started or Assigned in the Mailing List. $MTTR_{P_i}$ is median of the time required to fix bugs with Priority P_i (median is robust to outliers[6]). T_{total} is a normalizing parameter and is calculated as summation of the time taken by each participating bug.

5.4.3 *Bug Reporter Metrics*

In Issue Tracking System bugs are reported by various users (project members and non-project members). These users may report bugs with different quality. This fact was highlighted by Schugerl et al. in their work. They said that quality of reported bugs vary significantly [203] and good quality reported bug optimizes the time required to fix it and vice-versa [201].

Chromium in its bug reporting guidelines specified the factors that ensures the quality of the reported bugs. However, evaluating bug reports based on these guidelines is subjective [203] and so is assessing the contribution of the bug reporter (result of online survey mentioned in Section 5.4.1).

STATUS OF REPORTED BUG INDEX (SRI) A Closed bug may have status values (any one) defined in equation 9. The fraction of Closed bugs reported by a developer is a measure of contribution for the role of bug reporter. However, the status with which these bugs were Closed is a measure of quality of contribution. For instance, a bug Closed with status Fixed adds more value to the contribution of bug reporter than a bug closed as Invalid.

$$\boxed{SRI(r) = \frac{N_r}{N} \times \sum_{s=1}^{|s|} w_s \times \frac{C_s^r}{C_s}} \tag{8}$$

Let N be the total number of bugs reported and N_r be the total number of bugs reported by reporter r. w_s is a weight normalized tuning parameter defined for each C_s , where C_s is the count of bugs reported with status s. C_s^r is the count of bugs (reported by reporter r) with status s. s is defined as follows:

$$\begin{aligned} s = &< Fixed >, < Verified >, < FixUnreleased >, < Invalid >, < WontFix >, < Duplicate >, \\ &< ExternalDependence >, < Other^7 > \end{aligned} \tag{9}$$

The relative performance is a fraction calculated with respect to baseline S_{Mode}, where S_{Mode} is frequently delivered performance.

DEGREE OF CUSTOMER ECCENTRICITY (DCE) Google Chromium Issue Tracking System defines priority of a bug report in terms of the urgency of the task. A task is urgent if it affects business[8]. Among other factors, business is affected (measured by priority) if bugs in the software influences large fraction of the user base. Hence, reporting large fraction of high priority bugs is an indication of the degree of customer eccentricity.

$$\boxed{DCE(r) = \sum_{i=0}^{|P|} W_{P_i} \times \frac{N_{P_i}^r}{N_{P_i}}} \tag{10}$$

where W_{P_i} is a normalized tuning parameter with value proportional to the priority. N_{P_i} is the total number of priority P_i bugs reported. $N_{P_i}^r$ is the total number of priority P_i bugs reported by bug reporter r. This metric is defined for Closed bugs with status Fixed or Verified.

5.4.4 *Bug Triager Metrics*

In Triage Best Practices[9] written for Google Chromium project it is stated that identifying correct project owner is one of the roles of a bug triager. Guo et al. stated that determining ownership is

6 http://en.wikipedia.org/wiki/Outlier
8 http://sqa.fyicenter.com/art/Bug_Triage_Meeting.html
9 http://www.chromium.org/for-testers/bug-reporting-guidelines/triage-best-practices

one of the five primary reasons for bug report reassignment [204]. Jeong et al. in their study on Mozilla and Eclipse reported that 37%-44% of the reported bugs are reassigned which increases the time-to-correction [205].

A bug triager is supposed to conduct quality check on bug reports (to make sure that it is not spam and contains required information such as steps to reproduce etc.) and assign bug report to a developer who is (available) expert to resolve similar bugs. A bug triager should have good knowledge about the expertise, workload and availability of bug developers (fixers). Incorrectly assigned bug (error by bug triager) cause multiple reassignments which in turn increases the repair time (productivity loss and delays). We believe that computing the number of correct and incorrect reassignment is thus an important key performance indicator for the role of a bug triager.

Google Chromium in its Issue Tracking System does not give any account of the Triager who initially assigned the project owner but subsequent reassignments (through labels defined in comments in the Mailing List) are available.

REASSIGNMENT EFFORT INDEX (REI) Reassignments are not always harmful and can be the result of five primary reasons: identifying root cause, incorrect project owner, poor quality reported bug, difficulty to identify proper fix and workload balancing [204]. In an attempt to identify the correct project owner, triagers make large number of reassignments. These large number of reassignments to correctly identify project owner is a measure of the efforts incurred by a bug triager.

$$\boxed{REI(t) = \frac{N_t}{N} \times \sum_{\forall b} \frac{CR_b^t}{CR_b}} \tag{11}$$

Let N be the total number of bug reports which were at least once reassigned and N_t is the total number of bug reports which were at least once reassigned by triager t. This metric is defined for Closed bugs with status Fixed or Verified. For a given bug report b, CR_b is the count of total number of project owner reassignments and CR_b^t is the count of project owner reassignments by triager t.

ACCURACY OF REASSIGNMENT INDEX (ARI) Correctly identifying project owner is a challenging task. It involves large number of reassignments by multiple triagers. Each of these triagers make their contribution through reassignments. However, accuracy index (for the role of bug triager) is a measure of number of times triager correctly identifies project owner in a bug with large number of reassignments.

$$\boxed{ARI(t) = \frac{1}{n \times CR_{max}} \sum_{\forall b} CR_b} \tag{12}$$

where n is the total number of bug reports which were accurately reassigned (last reassigned) by triager t. For all bug reports b, CR_b measures the count of the total number of reassignments before it was accurately (finally) reassigned by triager t. CR_{max} measures the maximum number of reassignments before final. The value of ARI(t) lies between 0 and 1 (higher the value, more the contribution).

5.4.5 *Bug Collaborator Metrics*

A group of people who collectively (as a team) contribute to fix a bug are termed as bug collaborators. According to the user guide for Project Hosting Issue Tracker[10], collaborators provide additional information (in the form of comments in Mailing List) to fix bugs. These comments contribute in timely resolution of reported bug.

CUMULATIVE COMMENT SCORE (CCS) Introduction to Issue Tracker - a user guide for Project Hosting Issue Tracker states that developers make multiple comments in mailing list. The number of comments entered in mailing list is contribution of the developer and is used to measure performance.

$$\boxed{CCS(c) = \frac{1}{n} \sum_{\forall b} \frac{N_b^c}{N_b} * K_b} \tag{13}$$

10 http://code.google.com/p/support/wiki/IssueTracker

Here, N_b is the total number of comments in bug report b. N_b^c is the total number of comments entered by collaborator c in bug report b. n is the count of total number of bug reports. K_b is a tuning parameter to account for variable number of comments in bug reports. K_b is defined as ratio of N_b and N_{Mode}, where N_{Mode} is the count of number of comments entered frequently by collaborators (except collaborator c) on a bug b. This metric is defined for Closed bugs with status Fixed or Verified.

POTENTIAL CONTRIBUTOR INDEX (PCI) A developer can specialize and be an expert in one or multiple areas (as calculated in $SBI(d)$). Multiple developers (potential contributors) are added to CC-list based on prior knowledge (historical data or past experience) of their domain of expertise.

$$\boxed{PCI(c) = \sum_{\forall a} \frac{N_a^{cc}}{N_a} \times V_a} \tag{14}$$

Given a bug report b, PCI metric identifies list of contributors who are valued to fix it by assigning them values in the range from 0 to 1. It is defined for Closed bugs with status Fixed or Verified and CC-list defined. Here N_a is the total number of bugs reported in area a. N_a^{cc} is the number of times prospective collaborator c was mentioned in CC-list of bugs reported in area a. V_a is the importance of area of reported bug (as calculated in $PWFI$ for area a).

POTENTIAL CONTRIBUTOR INDEX-1 (PCI-1) $PCI-1$ extends the idea presented in PCI. In bug fixing lifecycle, contributors are incrementally appended to CC-list to meet additional support or expertise requirement.

$$\boxed{PCI-1(c) = \sum_{\forall a} \frac{N_a^{appCC}}{N_a} \times V_a} \tag{15}$$

Rest parameters being same (as explained in PCI), N_a^{appCC} is the count of number of times collaborator c was incrementally appended to the CC-list of bugs with Area a.

CONTRIBUTION INDEX (CI) When a bug is reported, developers are added to CC-list as potential or prospective contributors. However, only few people mentioned in CC-list contribute to fix bug through comments in Mailing List.

$$\boxed{CI(c,a) = \sum_{\forall a} \frac{N_a^{col}}{N_a^{cc}} \times V_a} \tag{16}$$

CI(c,a) measures expected to actual contribution for the role of bug reporter. Here N_a^{cc} is the number of times collaborator c was added in CC-list of bugs reported in area a. N_a^{col} is subset of N_a^{cc} in which developers mentioned in CC-list actually contributed to fix it. V_a is the importance of area of reported bug (as calculated in $PWFI$ for area a). It values contribution proportional to importance of area and normalizes it.

5.4.6 *Visualizations*

We use the general visualization model proposed by Zhang et al. [199]. We start by studying variegated sources for unconventional visualizations. We identify approximately 30 unique visualizations. For each visualization, we analyze the rationale behind its proposition and the nature of information that can be inferred from the visualization. We map these visualizations to the metrics to capture multiple information needs of decision makers. Each visualization technique maps to multiple metrics. However, due to limited space availability and for the ease of comprehension we present one metric for a visualization (refer Section 5.4.6). All visualizations present in this work are written in R language[11] (GNU project for statistical computing and graphics).

Unless specified otherwise all analysis is conducted on 'Closed' bug reports with status 'Fixed' or 'Verified'. We assign each developer a unique identity (chronologically). The unique identity is of the

11 http://www.r-project.org

form 'dxxxx' where 'x' is any numeric value. This unique identifier ensures anonymous behavior for ethical reasons and uniformity.

In this section we discuss 6 visualization techniques. Table 33 is an analysis and comparison of the 6 visualization techniques.

TRELLIS PLOT Trellis plot is a visualization technique to uncover relationships of variables in a multivariate dataset. Systematic application of Trellis plot shows how the response depends on explanatory variables [206][207]. We use Trellis plot visualization to analyze and interpret the performance of bug reporters using Status of Reported bug Index (SRI) metric. Figure 29 is an arrangement of rows, columns and panels. Panels in Figure 29 represent statuses of the bugs reported with state 'Closed'. They are arranged in the non-increasing order of importance of each status towards contribution and performance. Rows in each panel are the bug reporters arranged in non-increasing order of the number of bugs reported annually. Based on performance, bug reporters can be broadly categorized into five heads as 'Excellent', 'Very Good', 'Good', 'Satisfactory' or 'Non-Satisfactory'. The criteria for classification is specified in caption of Figure 29. The classification criteria of bug reporters can be fine-tuned to meet specific needs of organization.

In Figure 29, horizontal axis is fraction of bugs reported by bug reporter (shown on vertical axis in blue) for a given status (each panel) for two consecutive years 2009-10 and 2010-11 (represented by symbol variables: Red for the year 2009-10; Blue for the year 2010-11). Bug reporters arranged (top to bottom) in the non-increasing order of number of bugs reported followed by their score on Status of Reported bug Index (SRI) metric (in pink). 20 bug reporters are categorized into performance heads as 'Excellent', 'Very Good', 'Good', 'Satisfactory' and 'Non-Satisfactory' (based on the total number of bugs reported) ['Excellent' >250 bug reports, 'Very Good' >200 and <250 bug reports, 'Good' >100 and <200 bug reports, 'Satisfactory' >50 and <100 bug reports, and 'Non-Satisfactory' <50 bug reports]. Status arranged (top to bottom) in non-increasing order of relevance (in terms of contribution) for the role of bug reporter.

Following are interpretations for performance appraiser:

1. *Which bug reporters have high efficiency?* Efficiency is magnitude of time, effort or cost utilized to achieve target[12]. In Figure 29 we observe that bug reporter 'd28779' reports more bug reports than bug reporter 'd0001' (bug reporters arranged top to bottom in non-increasing order of total number of bugs reported). However, bug reporter 'd0001' has higher SRI score than bug reporter 'd28779' (as shown in pink on right). We conclude that bug reporter 'd28779' has higher efficiency than bug reporter 'd0001'. The explanation is that bug reporter 'd28779' places lots of efforts, however the usefulness of contribution is substantially less.

2. *How bug reporter's performance vary with time?* In Figure 29 we see that for large parts blue dots (fraction of bugs reported in year 2010-11) are ahead of red dots (2009-10). However, in panel 'Fixed/Verified' we see the two colored dots swapped and separated by large distances (w.r.t. other developers) for bug reporter 'd22575'. We infer that performance of bug reporter 'd22575' declined considerably in the year 2010-11 (w.r.t. year 2009-10).

Other interesting questions:

1. *Which bug reporters add overhead (in terms of resources) to the organization through their contribution?*

2. *What is the relative performance of a bug reporter?*

3. *Which bug reporter's issues receive less attention by team and hence get 'Closed' as 'IceBox'?*

4. *Which bug reporters report large fractions of 'Invalid' or 'Duplicate' issues?*

TREEMAP PLOT Treemap plot maps hierarchical information in a space-efficient and space-filling manner. It partitions the display space in rectangles. Each rectangle (sub-partition) is a child node with two distinguishing features: size and color. Size of a child node is proportional to the weight of sub-partition (w.r.t. whole) and color represents node-specific property. In a nutshell, Treemap encodes information as partitions (sub-partitions), size and color [208][209].

Treemap plot partitions rectangle into components (areas) namely BrowserUI, Misc, Internal, Webkit, and UI. Size of each component is proportional to the total number of bugs reported. Each

12 http://en.wikipedia.org/wiki/Efficiency

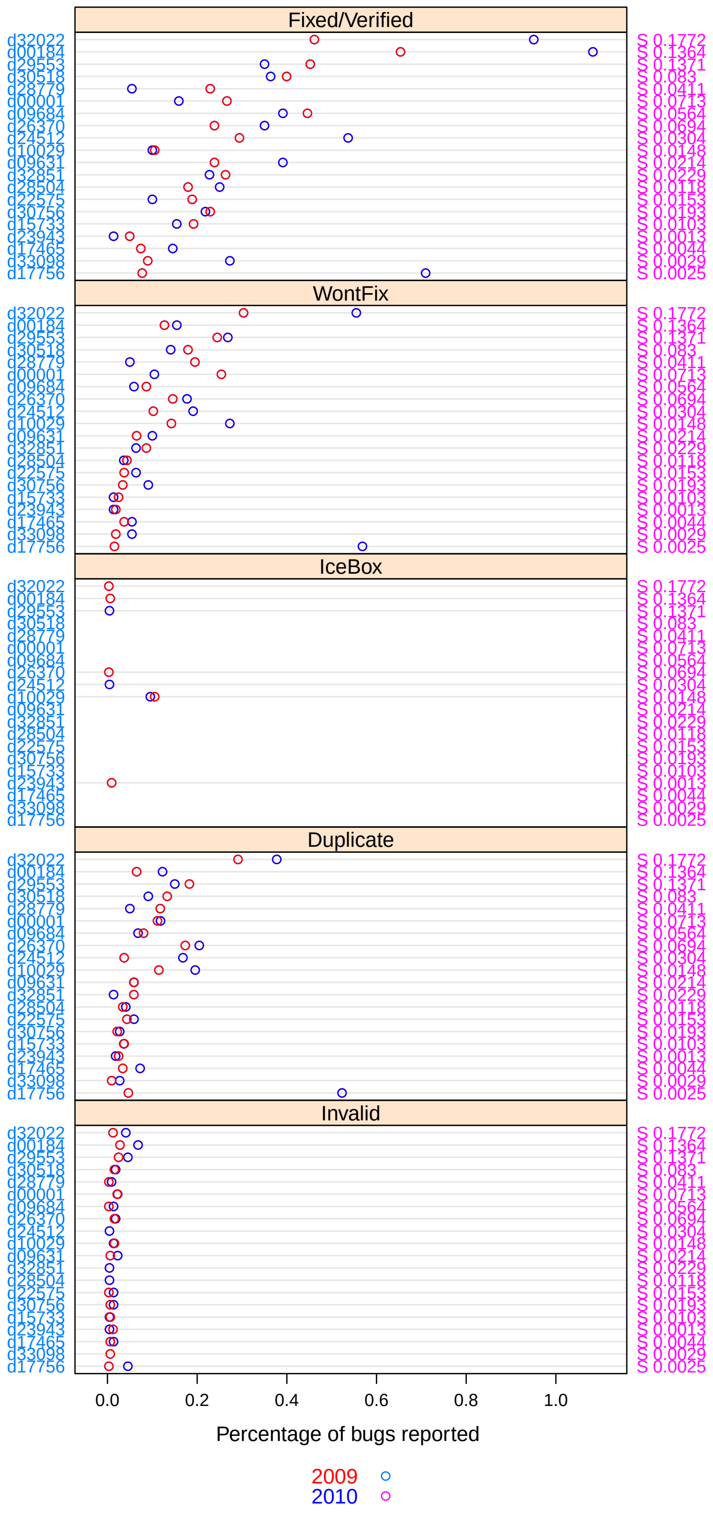


Figure 29: Trellis Plot

component is sub-partitioned on the bug collaborators, where size of sub-partition is proportional to the potential contribution index of developers for the component. The color of each sub-partition is an indicator of number of times bug collaborator actually contributed (on the scale above).

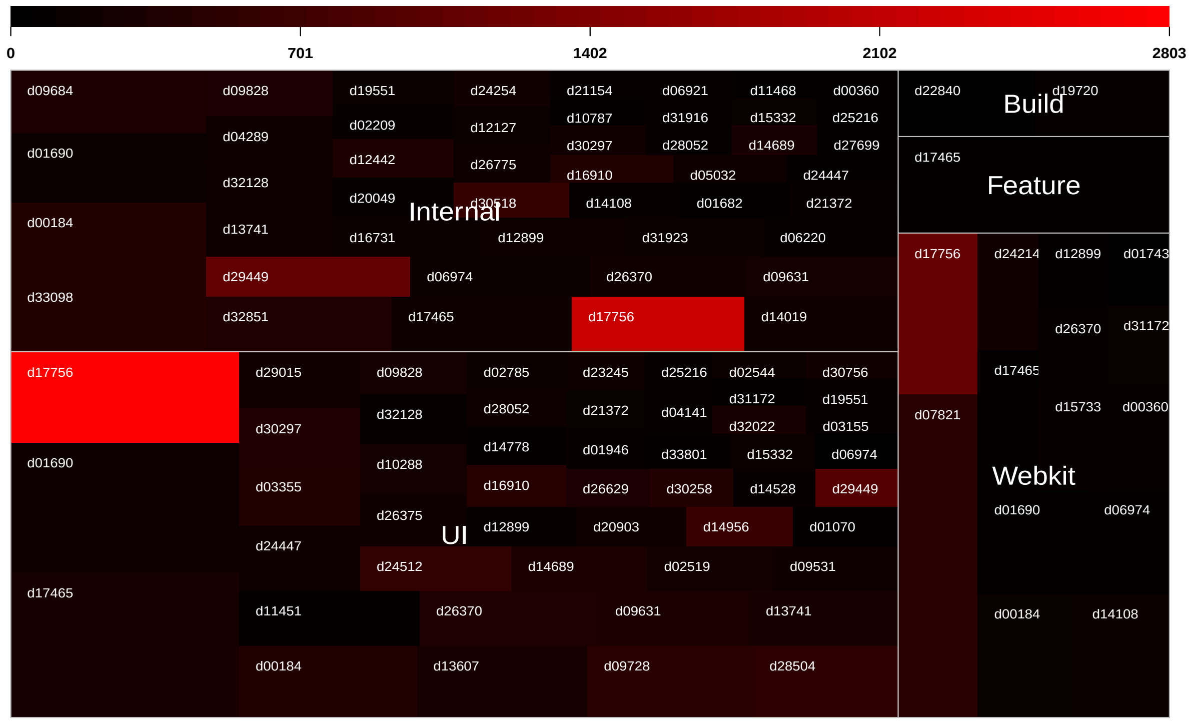


Figure 30: Treemap Plot

In Figure 30, Potential Contribution Index of bug collaborator is shown as the size of sub-partition of a component. Also, Contribution Index in Figure 30 is denoted by the color of sub-partition for each bug collaborator in Treemap. Color of sub-partition indicates initiative and contribution of bug collaborator. Detailed description in caption of Figure 30.

We present contribution of bug collaborators in 5 prominent components (areas in which maximum number of bugs were reported in year 2010-11, except area 'Undefined'). To ease comprehension in the limited space of the study, we include bug collaborators who contributed on more than 50 bug reports in an area for analysis. In Figure 30, bug collaborators are arranged bottom-to-top and left-to-right in the non-increasing order of Potential Contribution Index. Color-to-Size ratio indicates bug collaborators performance. High color-to-size ratio suggests performance more than expected (indicator of initiation and contribution) and vice-versa. Following are the inferences :

1. *Which bug collaborators are high performers?* In Figure 30, bug collaborator 'd17756' is distinguished from its neighborhood by contrasting color. Bug collaborator 'd17756' works significantly in three out of the five reported areas in Treemap and is an asset to the organization.

2. *Which bug collaborator is a critical resource to the organization?* In area 'Feature' of Treemap plot we see that only bug collaborator 'd17465' worked on more than 50 bug reports. Thus, bug collaborator 'd17465' is a critical resource to the organization.

3. *Which bug collaborators contribute more than expected?*

   In area 'UI' of Treemap plot we see that the size of sub-partition for bug collaborator 'd29449' is comparable to the size of sub-partitions for neighboring bug collaborators like 'd06974', 'd14528', etc. Figure 30 shows similar Potential Contribution Index for the above-mentioned bug collaborators. However, high color-to-size ratio for bug collaborator 'd29449' (compared to the neighbors) signifies more contribution than expected.

BERTIN'S HOTEL PLOT Bertin's Hotel plot is a homogenous structure that uses rearrangement of rows and columns to reveal information of interest. Input to the plot can be simple face values or complex derived (transformed) versions of the input. Global scaling is applied to the data to support homogenous nature [210]. Here, this visualization is applied on the Specialization and Breadth Index metric for the role of bug owner.

Specialization and breadth of knowledge are important for the role of bug owner and are related inversely. Issues with dependencies across components can be very well fixed by bug owners with some understanding of all linked components. Similarly issues which are deeply embeded in a component require bug owners with specialized knowledge in the area.

Bertin's Hotel plot is a matrix of areas and bug owners (detailed description in caption of Figure 31). Figure 31 shows the gross performing trends of bug owners for the year 2010-2011. The contribution of bug owners in these areas is range normalized (refer equation 17) to ensure homogeneity for comparison across areas.

$$RangeScore = \frac{x - min}{max - min} \qquad (17)$$

In Figure 31), horizontal axis shows the ten most popular components (areas) arranged chronologically in the non-increasing order of number of bugs reported in the area. Vertical axis is a list of 20 bug owners arranged in the non-increasing order of Specialization and Breadth Index (SBI) score for the year 2010-11. Contribution of bug owner in an area is measured relatively (w.r.t other bug owners working in the same area). Score scale is color coded. 100% imply distinguished contribution in an area and vice-versa. Shaded upper triangular matrix represents bug owners with breadth of knowledge and vice-versa.

In Figure 31 we see that the upper half triangle is a shade of gray. It implies that as we go from top to bottom, breadth of knowledge decreases and specialization increases. Following are the inferences relevant to appraisers:

1. *Which bug owners have specialized knowledge and vice-versa?* In Figure 31 we see that bug owner 'd09728' worked across multiple areas (except area 'BrowserBackend'). Also the collaboration pattern is almost uniform across areas (as shown by the slightly varying shades of gray). Thus, bug owner 'd09728' has breadth of knowledge. Similarly we see in Figure 31 that bug owner 'd02209' owns bug reports from two areas namely 'Internal' and 'BrowserBackend'. A striking contrast in the shades of gray in Figure 31 indicates that the bug owner 'd02209' has specialized knowledge in the area 'BrowserBackend'.

2. *Which bug owners are critical to my project?* For area 'ChromeFrame' in Bertin's Hotel plot, we see that bug owner 'd01682' owns maximum number of bug reports (dark shade of gray in column 'ChromeFrame' of Figure 31). Apart from bug owner 'd01682', no other bug owner contributed to the area 'ChromeFrame'. Thus bug owner 'd01682' with expert knowledge in 'ChromeFrame' is a critical resource to the organization.

3. *Which bug owner meets my project specific requirements?* Bug reports with high dependency across areas need bug owners with breadth of knowledge and vice-versa. For instance, projects in which bug reports have dependencies in 'UI' and 'Feature' will be best solved by bug owner 'd28504'. We see in Figure 31 that bug owner 'd28504' has good knowledge of area 'UI' and 'Feature' (as shown by the dark shades of gray for the two columns) and hence is a best fit for the project.

DART CHART Dart chart is a radial equivalent of the bullet chart. The name dart chart comes from its structural and conceptual resemblance to dart board. Bullet chart establishes a qualitative equivalent of performance from a quantitative measure of contribution score. The choice of classification scale is a function of the performance w.r.t. other developers or specific organization's expectations. Another aspect attached to this chart is performance relative to self [211]. Figure 32 is a visualization of Priority Weighted Fixed Index (PWFI) score for the role of the bug owner.

Understanding relationships between scores and its interpretation is a crucial task for organizations. However, such demarcations are not intuitive and are subjected to individual's perception. The aim of this plot is to justify performance ranking (classification) and ensure fairness and uniformity. Based on PWFI score, performance is broadly classified as 'Excellent', 'Very Good', 'Good', 'Satisfactory', and 'Not-Satisfactory' using min-max normalization of contribution of all bug owners for four halves in two consecutive years 2009-10 and 2010-11. Other normalization techniques may be applied to meet specific needs of organization.

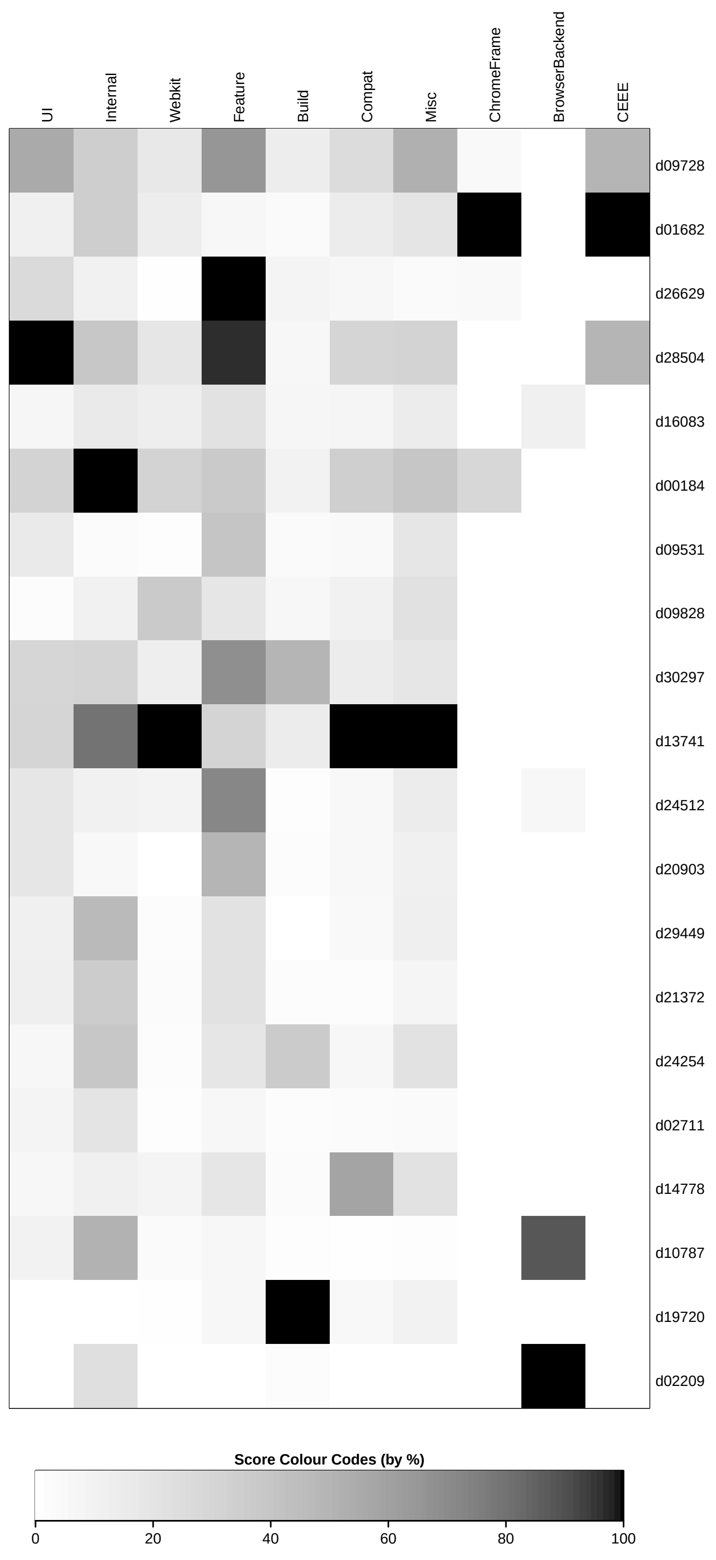


Figure 31: Bertin's Hotel Plot

In Figure 32, circle is divided into 50 sectors. Each sector is a bug owner arranged chronologically in the non-decreasing order (counter-clockwise) of Priority Weighted Fixed Issues (PWFI) score in the year 2010-11 (second half). Each concentric circle (outside to inside) is categorical classification of performance as 'Excellent', 'Very Good', 'Good', 'Satisfactory', or 'Not-Satisfactory'. This classification is based on quantitative score (PWFI score) ['Excellent': Score >0.5, 'Very Good': Score >0.25 and <0.5, 'Good': Score >0.125 and <0.25, 'Satisfactory': Score >0.0625 and <0.125, 'Not-Satisfactory': Score <0.0625]. Color of each concentric circle indicates total number of bug owners (as shown in legend 'count') that falls in a given performance category. 'Legend' shows performance in four halves over two consecutive years (2009-11).

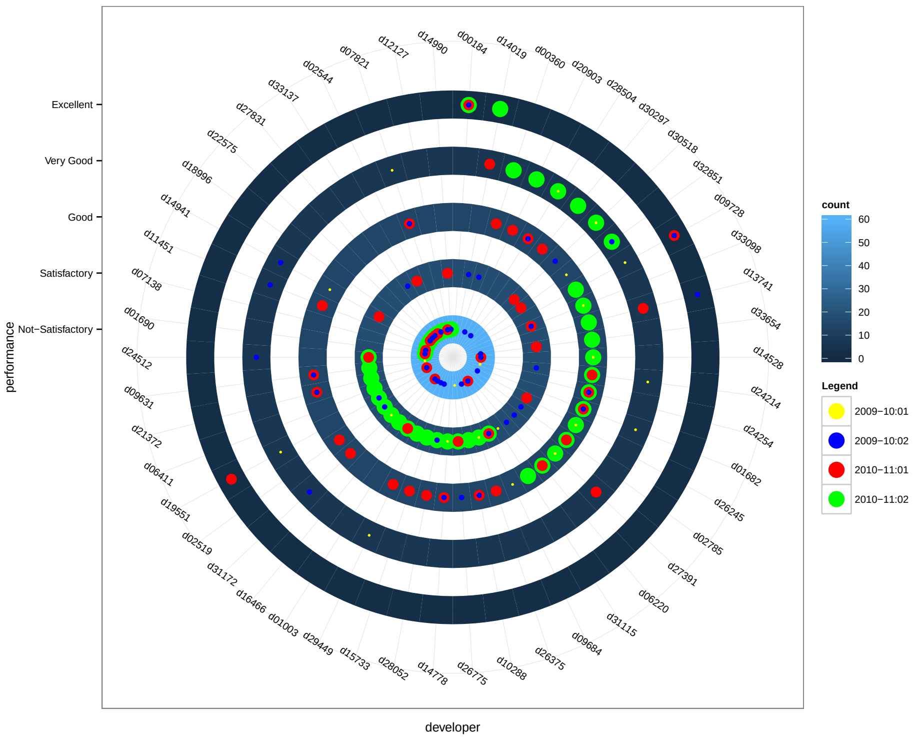


Figure 32: Dart Chart

Following are a few interpretations to help performance appraisers:

1. *Which bug owners are high performers relative to the team?* In Figure 32 we see color coded concentric circles. Lightest shade of innermost circle shows that a majority of bug owners deliver 'Not-Satisfactory' performance. Only two bug owners 'd00184' and 'd14019' achieve 'Excellent' performance for their outstanding contribution in second half of 2010-11 (shown as green colored dots on outermost circle).

2. *How consistent is the performance of bug owner?* In Figure 32 we observe three overlapping dots for bug owner 'd00184' in the outermost circle. These three dots are for the three halves: green for the second half of 2010-11, red for the first half of 2010-11 and blue for the second half of 2009-10. The yellow dot for the bug owner 'd00184' lies in the innermost circle (i.e. 'Not-Satisfactory' performance for the first half of 2009-10). Thus bug owner 'd00184' starts with 'Not-Satisfactory' performance and shows marked improvement by consistently performing 'Excellent' for the next three halves. The positions of dots for bug owner 'd19551' in Figure 32 shows periodic fluctuations in performance. Bug owner 'd19551' performs well in the first halves of the two consecutive years, while in the second half performance observes a steep fall. Similarly, bug owners with marked improvement, slight improvement, consistently average, steep decline in performance, etc. can be observed.

HYBRID PLOT Hybrid plot is a combination of simple plots to display rich and otherwise complex information. Adjacent plots share axis to link different pieces of information and present it as a whole [212]. In this plot, we study Deviation from Median Time To Repair metric for the role of bug owner.

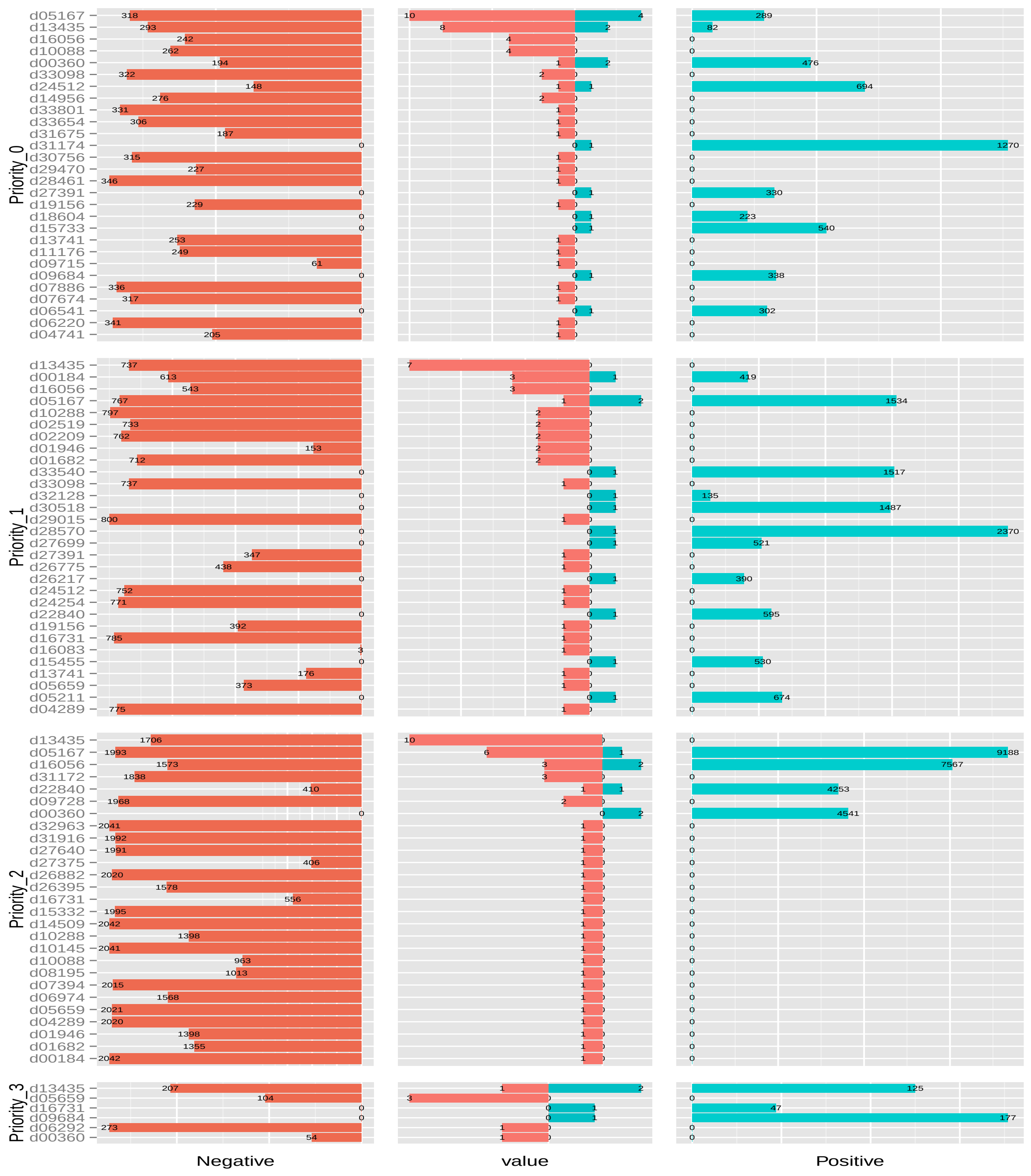


Figure 33: Hybrid Plot

Figure 33 consolidates twelve plots of two types: back-to-back bar chart and simple bar chart. The arrangement of graphs is such that the adjacent plots share axis and present same information. Vertical axis of the plot has two levels of hierarchy in information. At level one, information is presented for all priorities i.e. priority 0, 1, 2 and 3. For each priority, we analyze the contribution of bug owners arranged in the non-increasing order of the total number of bugs owned.

In Figure 33 we study 'Security' bug reports 'Closed' by bug owners during 2010-11 (first half). The time to fix is the minimum time between bug 'Start' or 'Assigned' status and 'Fixed' or 'Verified' status. Cases in which multiple bug owner reassignments occur are not considered. The choice of Expected Time To Repair for a given priority bug can be organization specific.

Based on priority of bug reports vertical section is divided into four segments as P0, P1, P2 and P3 (P1 is highest; P3 is lowest). Each segment in the vertical section is a list of bug owners arranged in the non-increasing order of total number of security bug reports owned. Horizontal axis measures three different scores. 'Value' is count of bug reports owned by a bug owner for a given priority. Red and blue colored 'Value' is the count of bug reports where bug owner exceeded/preceded the time specified respectively. Similarly, 'Negative' and 'Positive' shows the time exceeded/preceded (in hours) to fix the bug report (w.r.t. expected time) respectively.

Following are the possible inferences for decision makers:

1. *Which bug owners have expertise in solving a given priority 'Security' bug report?* In Figure 33 we see that for each priority, bug owners are arranged (top to bottom) in the non-increasing order of the value of back-to-back bar chart. Back-to-back bar chart shows the total number of bug reports owned with a given priority. Red bar in the back-to-back bar chart measures the number of bug reports with negative contribution and vice-versa.

   In Priority P0 panel of Hybrid plot, we see that bug owner 'd05167' owns maximum number of bug reports and thus has expertise in solving priority P0 bug reports. Similarly, we can derive inferences for other priorities.

2. *Which bug owner delivers good management skills?* In Priority P0 panel, we observe that 'Positive' bar chart has maximum value for bug owner 'd31174'. However this maximum value is for one bug report (shown as blue color bar of the back-to-back bar chart). Thus bug owner 'd31174' shows good management skills. However, one bug report does not establish the desired confidence in result. Combining these two factors, we infer that bug owner 'd05167' has good management skills (worked on four bug reports and 'Fixed' them 289 hours before expected time). Similarly, we can infer bad management skills.

NIGHTINGALE ROSE PLOT Nightingale Rose plot (also known as coxcomb) is a radial plot. It analyzes the gap between the actual value and the corresponding reference expected value. Nightingale Rose plot displays contrast by analyzing data for two consecutive years. However to ensure comparisons and to analyze sophisticated interactions between causative factors the period i.e. time interval between two consecutive observation points must be uniform [213][214]. For this visualization, we use Priority Weighted Fixed Issue metric to gain detailed insights on performance trends.

In Figure 34, each sector of the circle is a month of the year. Two radial plots are drawn presenting data for two consecutive years (Regime:Year 1 for 2009 and Regime:Year 2 for 2010). Bug owners are represented by different colors as shown in the 'Legend'. Area of wedge shows the performance of bug owner (for a given month of the year) where radius of wedge is square root of the Priority Weighted Fixed Issue (PWFI) score. Bug owners are arranged (inside to outside) in the non-decreasing order of their PWFI Score.

Figure 34 is an activity track of 10 bug owners for two consecutive years (2009-10 and 2010-11). The selection of these bug owners is based on their PWFI score (detailed description in caption of Figure 34). The inferential power of Nightingale Rose plot can be harnessed by capturing environmental factors. In the absence of the information related to the environmental factors, the inferences stated below are exemplary and must be put into right perspective for analysis by the organizations.

1. *Which bug owners are environmental susceptible? or Which bug owners can act as a savior for the team?* In Figure 34 we see a steep decline in the quality contribution for the month of December (w.r.t. the neighbouring months November and January) for years 2009-10 and 2010-11. One possible justification is reduced activity during Christmas vacations. However, we observe that bug owner 'd05167' contributes significantly in times of general inactivity.

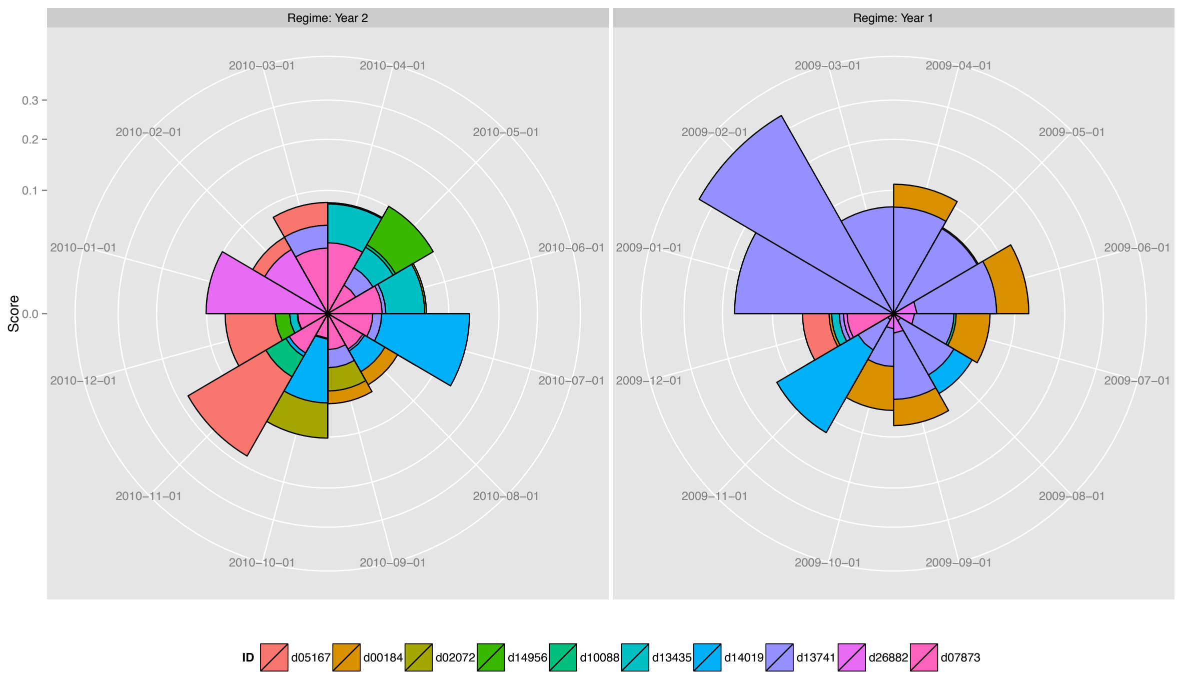


Figure 34: Nightingale Rose Plot

2. *Analyze performance variations within/across years?* In Figure 34 we see that the contribution of bug owner 'd07873' was overshadowed (wedge disappear) by the significantly large contributions by other bug owners. However, for the year 2010 we observe that the wedge corresponding to the bug owner 'd07873' appears for all months of the year. It shows quality contribution delivered significantly.

3. *Which bug owner's contribution dominate in a given time period?* Assume organization has product release scheduled in April 2009. In times of peak workload we observe that the contributions of bug owner 'd13741' and 'd00184' overshadow the contribution of other bug owners. These bug owners deliver quality contribution to ensure timely release and are asset to the organization.

Table 33 summarizes the 6 visualization techniques and compare them.

Table 33: Analysis and comparison of visualizations

Features	Trellis Plot	Treemap Plot	Bertin's Hotel Plot	Dart Chart	Hybrid Plot	Nightingale Rose Plot
Metrics	SRI	PCI,PCI-1,CI	SBI	PWFI	DMTTR	PWFI
Data Dim.	5	3	3	4	5	3
Approx. Team Size	15-20 members	50-75 members	25-50 members	50-100 members	15-20 members	15-20 members
Appl	Study trends and outlier behavior by systematic arrangement of variables; Temporal analysis of performance; Measure performance relatively	Map data hierarchically; Panoramic view to individual's contribution in a team; Compare and contrast performance relatively	Pattern analysis from rearrangement of rows and columns based on criteria	Establish qualitative equivalent of quantitative score; Analyze temporal variations in performance; Combine statistical data for each class of observations	Study positive and negative contribution	Impact of environmental factors on performance; Compare actual vs expected

Table 34: Survey Results [PoR=Percentage of Relevance; CPoR=Cumulative Percentage of Relevance]

Metrics	RQ	Not at all Effective(1)	Slightly Effective(2)	Moderately Effective(3)	Very Effective(4)	Extremely Effective(5)	Total	Mode	PoR	CPoR
Dart Chart	RQ1	0	0	3	5	0	8	4	100	100
	RQ2	0	0	3	2	2	7	3,4	100	
Bertin's Hotel	RQ1	0	2	4	4	0	10	3,4	80	83.33
	RQ2	0	1	3	6	0	10	4	90	
	RQ3	0	2	5	3	0	10	3	80	
Treemap Plot	RQ1	0	2	3	5	0	10	4	80	73.33
	RQ2	1	2	4	2	1	10	3	70	
	RQ3	1	2	3	4	0	10	4	70	
Trellis Plot	RQ1	2	1	2	2	1	8	3,4	62.5	75
	RQ2	1	0	4	3	0	8	3	87.5	
	RQ3	1	1	4	1	1	8	3	75	
	RQ4	1	1	3	3	0	8	3	75	
Hybrid Plot	RQ1	0	1	1	4	1	7	4	85.71	92.85
	RQ2	0	0	2	3	2	7	4	100	
Nightingale Rose	RQ1	0	2	4	1	0	7	3	71.43	66.96
	RQ2	1	2	1	4	0	8	4	62.5	

5.4.7 *Evaluation*

Post-implementation, we conducted a survey of practitioners. We received 10 valid responses from the practitioners in a large global IT industry. 9 survey respondents had more than 5 years of experience and 1 had more than 1 year of experience. 8 survey respondents were Project Managers, 1 was head of R & D initiatives and 2 were Bug Owner/Bug Fixer. The roles were overlapping in nature. 8 out of 10 survey respondents had been appraisers in the past or present.

Table 34 shows the usefulness of the visualization techniques when compared with the existing practices in organization to assess contribution and performance. Research questions (RQ) asked for each visualization technique are arranged in order of reference in the study.

Percentage of Relevance (PoR) is the percentage of responses which consider RQ effectively answered by visualization technique where relevant responses are those with value 'Moderately Effective' or higher. Similarly, Cumulative PoR (CPoR) is mean of all RQs for a visualization technique. Survey results show that the usefulness of visualization techniques vary considerably. However, general consensus, as suggested by the mode of survey results, validates the usefulness of our work.

5.5 MEASURING CHANGES IN CONTRIBUTOR COMMUNITY PARTICIPATION

We investigate the stability of community in software maintenance projects by mining Issue Tracking System. We probe temporal community contribution patterns to analyze trends and estimate future participation to support planning and decision making. We study literature in community management to understand stability indicators. We identify three KSIs namely attrition, regeneration, and retention. We propose metrics to measure the KSIs and compute their time series data. We explore the inferential ability of the time series data through research questions that may help project managers and decision makers in informed decision making.

5.5.1 *Attrition Rate*

During the maintenance phase of software development lifecycle projects loose its critical human resource also termed as Attrition. These contributors may be regenerated (substituted) or lost completely [52]. Attrition Rate (AR) for time interval T_t is the percentage of contributors who left the project in time interval T_t to the total number of contributors who participated in the preceding time interval T_{t-1}. If the contributors do not participate for one time-interval we assume that they have left the project. In set notations, the attrition rate is the cardinality of set difference[13] of contributors (Con) in two consecutive time intervals T_{t-1} and T_t divided by the cardinality of all participant contributors in the time interval T_{t-1} multiplied by 100 (refer Equation 21). The unit of Attrition Rate is $quarter^{-1}$.

$$\boxed{|AR_{T_t}| = \frac{|Con_{T_{t-1}} \setminus Con_{T_t}|}{|Con_{T_{t-1}}|} \times 100 \qquad where\ t > 1} \tag{18}$$

5.5.2 *Regeneration Rate*

FLOSS projects evolve temporally with contributors entering or leaving the project at will. Research shows that new entrants drive a large fraction of open source projects as original contributors leave the project [195]. Regeneration replenishes the resources lost due to contributor attrition and help meet additional resource requirement in the project. However, a very high regeneration rate may indicate problems with the team composition. For instance, high regeneration rate indicates imbalanced team composition (junior to senior ratio) with smaller fractions of experienced contributors. The imbalanced team composition may adversely influence the stability of the project.

Regeneration Rate (RgR) for time interval T_t measures the rate (in percentage) at which new contributors start participating in ITS in two consecutive time intervals under analysis. In set notations, we define regeneration rate for time interval T_t as fraction of cardinality of set difference of contributors (Con) in T_t from contributors (Con) in time interval T_{t-1} to the cardinality of contributors (Con) in time interval T_t multiplied by 100. The unit of measurement is $quarter^{-1}$.

$$\boxed{|RgR_{T_t}| = \frac{|Con_{T_t} \setminus Con_{T_{t-1}}|}{|Con_{T_t}|} \times 100 \qquad where\ t > 1} \tag{19}$$

5.5.3 *Retention Rate*

Knowledge acquired with experience, formal and informal understanding of the project cannot be transferred [63]. When a contributor leaves project, knowledge acquired goes with the contributor causing knowledge loss. A key parameter to understand the stability of the software maintenance project is to know its ability to retain knowledge or its contributors.

Retention Rate measures contributors retained by the project. Retention Rate (RtR) for time interval T_t measures percentage of contributors retained out of all participants in time interval T_{t-1}. In set notations, Retention Rate for time interval T_t is the ratio of cardinality of intersection of contributors (Con) in time interval T_t and preceding time interval T_{t-1} to the cardinality of contributors (Con) in time interval T_{t-1}. The unit of Retention Rate is $quarter^{-1}$.

$$\boxed{|RtR_{T_t}| = \frac{|Con_{T_t} \cap Con_{T_{t-1}}|}{|Con_{T_{t-1}}|} \times 100 \qquad where\ t > 1} \tag{20}$$

The three metrics proposed in the study cannot be compared. So to help cross comparison, we normalize the metrics on the union of contributor count in two consecutive time intervals T_t and T_{t-1}. The normalization facilitates comparison at the cost of affecting the scores. In this study we conduct analysis of modified metrics to ease comparison. The three metrics can be restated as:

$$\boxed{|AR_{T_t}| = \frac{|Con_{T_{t-1}} \setminus Con_{T_t}|}{|Con_{T_{t-1}} \cup Con_{T_t}|} \times 100 \qquad where\ t > 1} \tag{21}$$

$$\boxed{|RgR_{T_t}| = \frac{|Con_{T_t} \setminus Con_{T_{t-1}}|}{|Con_{T_{t-1}} \cup Con_{T_t}|} \times 100 \qquad where\ t > 1} \tag{22}$$

13 Set difference is set of all contributors who worked in one time interval (T_{t-1}) but did not continue participation in next time interval (T_t)

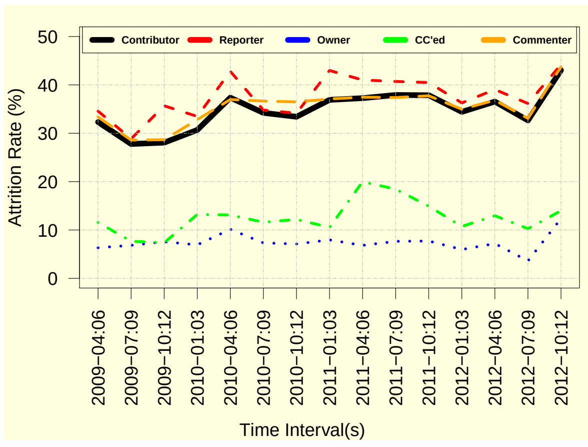


Figure 35: Attrition Rate of contributors across four years

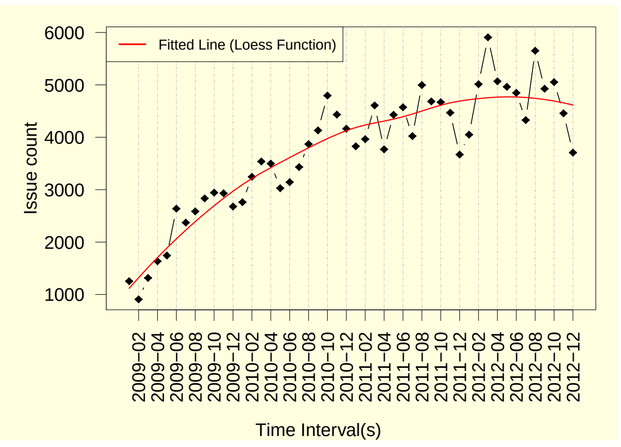


Figure 36: Issue count distribution calculated monthly

$$\boxed{|RtR_{T_t}| = \frac{|Con_{T_t} \cap Con_{T_{t-1}}|}{|Con_{T_{t-1}} \cup Con_{T_t}|} \times 100 \qquad where\ t > 1} \tag{23}$$

Together the three metrics sum to 100 as shown below:

$$\boxed{AR + RgR + RtR = 100} \tag{24}$$

Figure 35, Figure 37, and Figure 38 show Attrition Rate, Regeneration Rate, and Retention Rate respectively for 15 time intervals starting from '2009-04:06'. The horizontal axis of the plot shows consecutive time intervals measured quarterly and the vertical axis shows metric score in percentage. Colored lines (refer to the legend) shows metric scores for contributors and the four roles of contributors (reporter, owner, cc'ed and commenter).

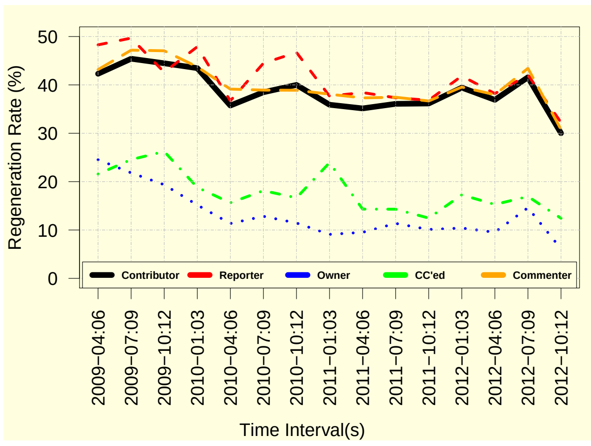
5.5.4 *Applications*

RQ1: Do we observe high Attrition Rate in Google Chromium Project?
In Figure 35, contributor Attrition Rate in Google Chromium Project (shown in black) ranges from 27% to 47% with 34.7% mean and 4.07 % standard deviations. It implies that every three months one-third of the contributors discontinue to participate in ITS. The high attrition rate in GC-ITS raises concerns regarding the stability of the project. However, it fails to give a complete picture of the status of the project. High Attrition Rate makes it relevant to conduct further analysis to understand its cause and hence RQ2.

RQ2: Do we observe comparable Attrition Rates for all roles? If no, how does it vary with role relevance?
In Figure 35, we observe a marked difference in the Attrition Rates for the four roles where role relevance follows the order: owner $\geqslant$ cc'ed $\geqslant$ commenter $\geqslant$ reporter. We observe that Attrition Rate increases with decreasing relevance of the role. We see minimum Attrition Rate for owner (shown in blue) and maximum for reporter (shown in red). This observation follows the intuition and results of previous research. Attrition Rate for reporter varies from 28.9% to 44.2% with mean 37.7% and standard deviation 4.3%.

The issue count reported in GC-ITS increases over time (refer Figure 36). Thus a high Attrition Rate for the role of reporter presents an open question that is, whether the project is able to compensate for the resources lost and generate resources to meet the increase in requirements. Subsequent RQs address these concerns.

RQ3: Is high Attrition Rate accompanied by high Regeneration Rate?



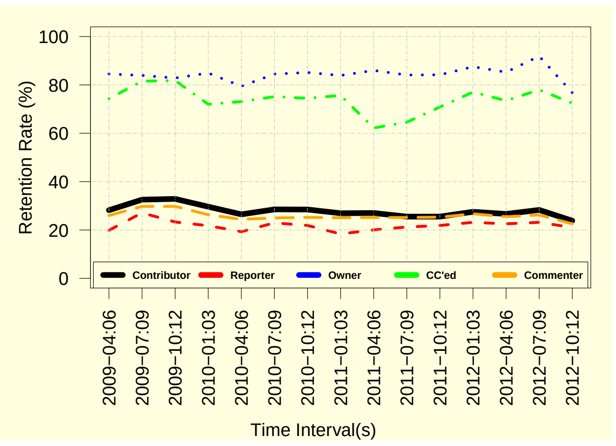


Figure 37: Regeneration Rate of contributors across four years

Figure 38: Retention Rate of contributors across four years

In Figure 37, we see that in GC-ITS contributor Regeneration Rate fluctuates from 30.0% to 45.4% with mean 38.7% and 4.2% standard deviation. This indicates that the resource lost is regenerated. However, like RQ2, do we observe a marked difference in regeneration rates for four roles and how it may impact the stability. Our next RQ attends anomalous behavior, if any, in Attrition Rate and Regeneration Rate patterns for four roles.

RQ4: Do we observe anomaly, if any, in the temporal behavior of Attrition Rate and Regeneration Rate for the four roles of contributors?
In Figure 39, we present contribution patterns for the four roles of contributors. Dotted lines represent actual Attrition Rate and Regeneration Rate while dark lines indicate smoothed data (moving average with past 3 observations) to generate approximate function to capture important patterns. The horizontal axis shows the time intervals while the vertical axis shows the scale in percentage to measure Attrition Rate and Regeneration Rate. Colors in the legend specify the values being measured by dotted or dark line. In Figure 39 we observe that for the owner Regeneration Rate is always higher than the Attrition Rate while the same does not hold true for the other three roles. For reporter, commenter, and cc'ed we observe cross overs and fluctuations. Cross overs and fluctuations may point to concerns. Thus, in the next RQ we analyze what temporal trends signals change in the stability of GC-ITS.

RQ5: Do we observe changes in lag between Attrition Rate and Regeneration Rate?
In Figure 39, we see that for owners, Attrition Rate remains low and consistent for most parts while a constant dip in Regeneration Rate is observed. Similarly for other roles we observe decrease in lag between Attrition Rate and Regeneration Rate accompanied by crossovers and fluctuations. Thus, in the last few time intervals contributors lost due to high Attrition Rate are not replenished with high Regeneration Rate. Also, though we do not observe this trend for owner their participation pattern is also heading on similar lines.

RQ6: Do we observe increasing or decreasing trend in Retention Rate over time?
Figure 38 shows that contributor Retention Rate ranges from 23% to 32% with wide variations across four activities. Retention Rate is high for owner and cc'ed compared to the retention rate for reporter and commenter. One observation from this graph is that commenters stay in projects longer than the contributors who report bugs.

RQ7: Do we observe changes in the relationship between Attrition Rate, Regeneration Rate, and Retention Rate? What implications can we draw from such observations?
To answer this RQ, we approximate the trend by fitting a linear model to the temporal data of Attrition Rate, Regeneration Rate, and Retention Rate for 15 time intervals. The simple linear regression model creates an approximation of the trend with correlation coefficients r of 0.73, -0.71, and -0.70 respectively. The fitted model is statistically significant with p-values 0.005, 0.008 and 0.004 respectively. The fitted model provides an approximation of the trend. Moreover, the residuals of the model have mean close of zero and a finite variance indicating that the residuals do not follow any pattern.

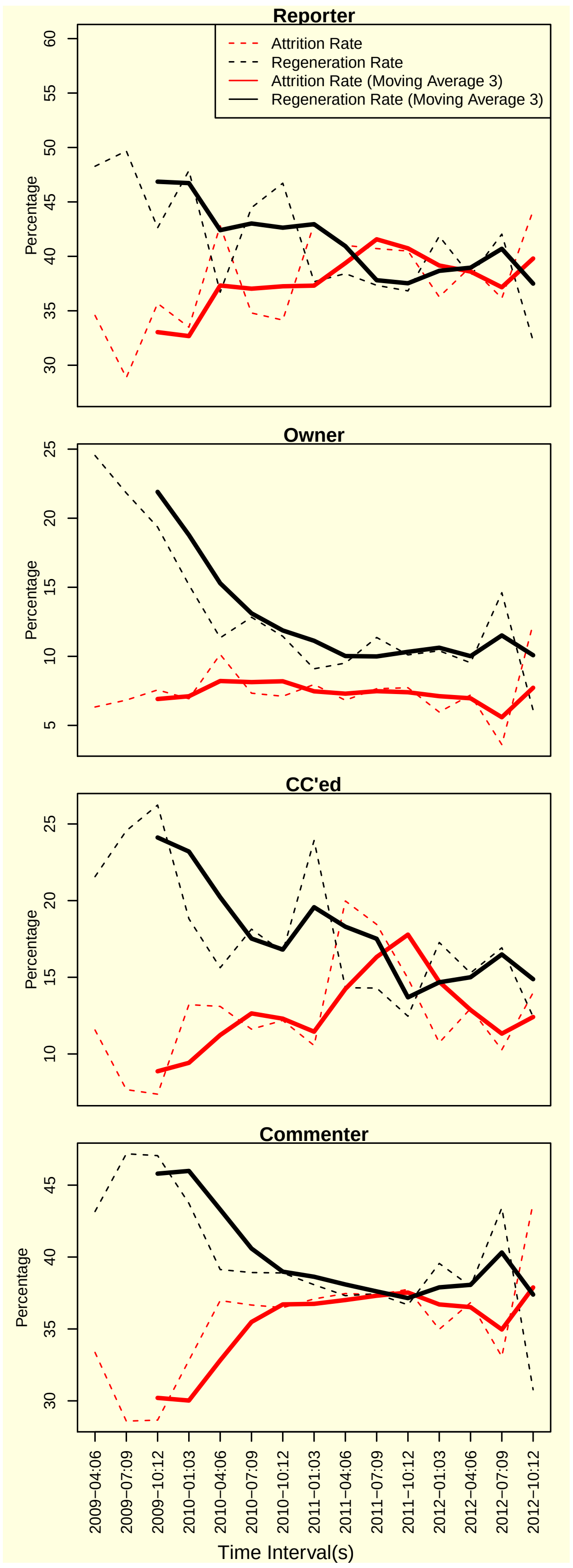


Figure 39: Comparison of Attrition Rate and Regeneration Rate of contributors across four years

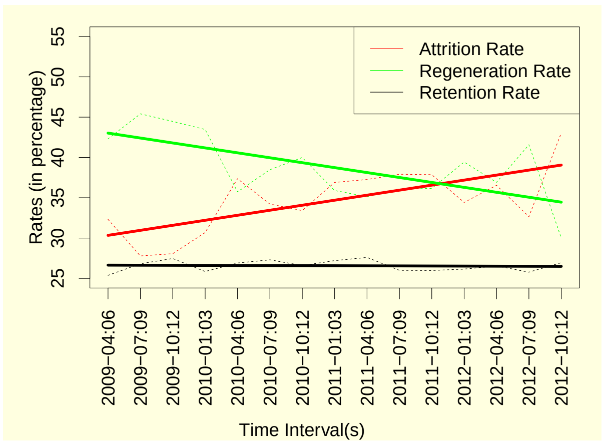


Figure 40: Changes in contributor participation pattern

In Figure 40, we observe that Regeneration Rate is decreasing, Attrition Rate is increasing, and Retention Rate is decreasing. Thus over time contributors leaving the project are not regenerated and the retention of contributors is decreasing over time. In a nutshell, the contributors are slowly moving away from the project. This observation may influence the stability of the project depending on whether the project is in initial phase or has attained maturity. Decreasing contributor participation during initial phase may indicate decrease in interest and hence a decrease in the popularity of the project. While if we observe the same trend once the project is mature, it may indicate that the project observes few issues that must be incorporated. This observation leads us to the last RQ that examines the influence of issue counts on participation.

RQ8: Do we observe an increase in contribution overhead on each contributor with time?
In Table 26 we observe an increase in the contributor count with increase in issue count. However, the increase is not proportional. Thus over time there is an increase in workload on contributors. The increase in workload may be welcoming or may further lead to decrease in contributor participation.

ESTIMATING FUTURE PARTICIPATION Planning is the response of forecasting. Estimating future participation helps project managers plan ahead and facilitate informed decision making. Research in FLOSS projects shows that contributor participation is a volunteering activity where contributors join or leave project at will. Individual participation or contribution at microscopic level cannot be estimated. However, participation at the macroscopic level follows trend [215].

In this section, we use historical data to make short-term predictions of contributor participation patterns. The aim is to explore the time series data of KSI metrics for trends and investigate the confidence in predicting future participation. We investigate the correlation between the community participation pattern and stability of the community. Since we model historical participation data, the study does not capture changes in participation pattern due to changes in external factors like similar project gaining popularity, etc. Though these changes will get reflected in subsequent time intervals.

We propose statistical forecasting models that along with judgmental forecasting of decision makers (to capture environmental factors) will present a complete understanding of factors that influences the stability of the project. We conduct experiments on GC-ITS dataset for 15 time intervals. We enumerate three statistical prediction models, justify the choice, implement, compute accuracy, compare the models for goodness of fit and forecast accuracy and present visual analysis. We present results for the three metrics and visualization of one metric. Given the small data size we choose 12 (out of 15) data points for training and the rest 3 for testing. The small dataset influences the selection of modeling technique as discussed below. However, it has limited influence on the accuracy of the prediction as old time series data is less useful to study changes in the project. All experiments present in this section are implemented using toolkits available in R language[14].

The baseline model (Model-I) for prediction assumes (also conducted experimentally) that the contributor participation pattern follows normal distribution. We compute mean and standard deviation of the time series data with 99% and 95% confidence interval to estimate participation patterns in following three time intervals (refer Table 35). In Table 35 we see that the simple statistical model

14 http://www.r-project.org

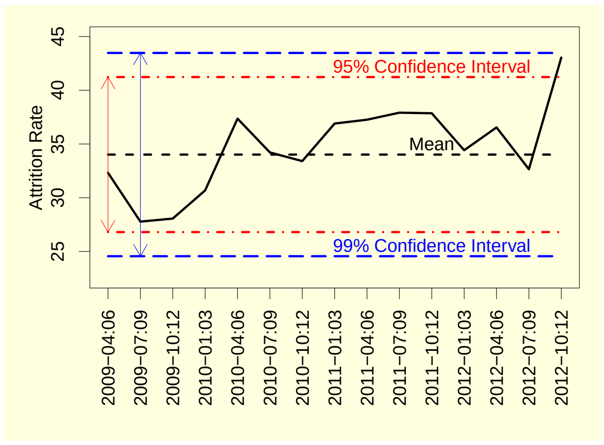


Figure 41: Model I: Simple Statistical Model to analyze trends in time series data of Attrition Rate

Table 35: Model-I: Simple Statistical Model [SD=Standard Deviations; Conf=Confidence Interval; Obs=Observations for next three time intervals]

Metrics	Mean	SD	99% Conf	95% Conf	Obs
AR	34.02	3.68	24.56-43.47	26.80-41.23	36.54, 32.65, 43.04
RgR	39.38	3.73	29.78- 48.97	32.06-46.70	36.54, 32.65, 43.04
RtR	28.27	2.40	22.10- 34.44	23.57-32.97	26.64, 28.29, 23.84

computes future participation with 99% confidence interval. Figure 41 show the simple statistical model and the confidence interval for Attrition Rate, Regeneration Rate, and Retention Rate metrics respectively.

The baseline model sets benchmark for the two more involved statistical models. To select models, we conduct exploratory analysis to understand the composition of time-series data. In Figure 42, we decompose the time-series data into trends, seasonal components, and remainders using *stl* function in R. *stl* function in R uses loess smoothing to identify seasonal, trend, and irregular components. We use the default settings with seasonality set to periodic. A preliminary analysis of time series data of Attrition Rate metric (also implemented for Regeneration Rate and Retention Rate metrics) suggests that the seasonality (if any) does not grow with the trend. Thus we use an additive model to represent time series data. Figure 42 shows additive components of the time series data along with actual observed data. In Figure 42, we see that Attrition Rate observes an increasing trend with the increase in the irregular component or remainder. The increase in the irregular component with time indicates that the time series does not observe seasonality though we may observe cycles with the trend. In Figure 42 we observe that the time series data of Attrition Rate metric increases linearly in time. Thus, the model close to the observed trend is a linear regression model. We model the time series data of KSI metrics using *tslm* function. We estimate the goodness of fit of the model and predict accuracy and confidence interval of the observations. The code snippet below from R shows the goodness of fit of the model.

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  29.1656     1.6195  18.009 5.97e-09
x             0.7464     0.2201   3.392  0.00686

Residual standard error:
2.631 on 10 degrees of freedom
Multiple R-squared: 0.535,
Adjusted R-squared: 0.4885
F-statistic:  11.5 on 1 and 10 DF,
p-value: 0.006864
```

From statistical significance testing, p-value less than 0.01 indicates that the linear fit provides a good estimation of the trend. We then use the fitted model to measure the predictive ability of the model with 80% and 95% confidence intervals. Figure 43 show the observed trend, fitted model, and predicted observations along with the confidence intervals. We see in Figure 43 that the fitted

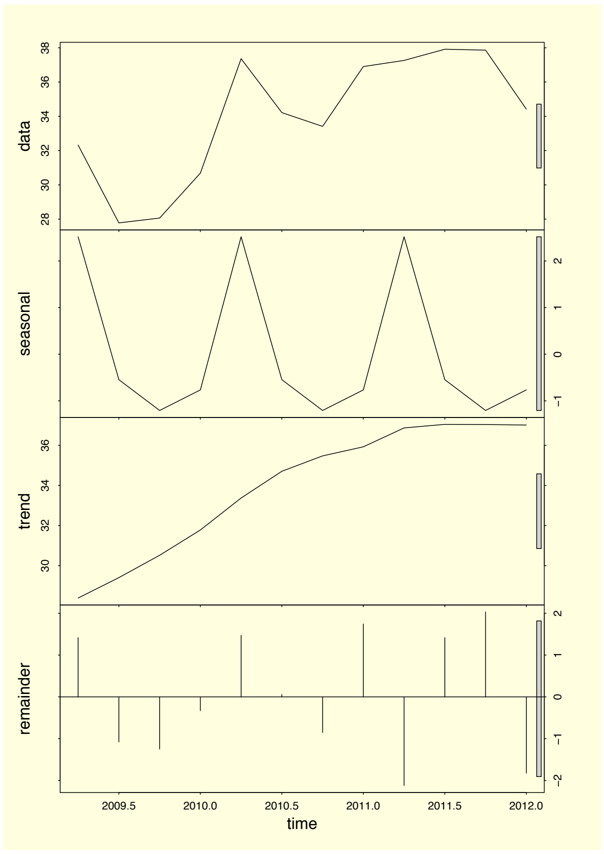


Figure 42: Decomposition of time series data of Attrition Rate metric into trend, seasonal and remainder component using stl function in R

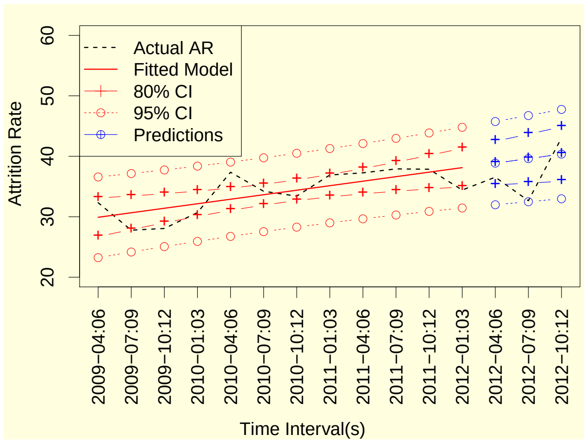


Figure 43: Model II: Linear Regression Model to analyze trends in time series data of Attrition Rate

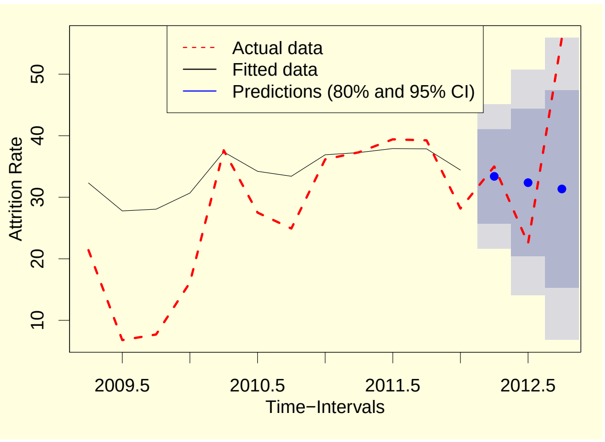


Figure 44: Model III: Exponential Smoothing Forecasting to analyze trend in time series data of Attrition Rate

model predicts Attrition Rate in next three time intervals with 95% confidence. We observe that the participation varies substantially from the fitted model. In the next model we try to model a better fit to the data (without overfitting) to address this concern.

For statistical time series modeling, exponential smoothing and ARIMA models are two popular techniques. The small dataset available for prediction makes estimating parameters for ARIMA modeling difficult without overfitting. Thus ruling out the choice of ARIMA modeling for the study. Exponential smoothing, on the other hand, uses recent observations for prediction. Therefore, for third model we choose exponential smoothing to predict contributor participation patterns. To implement Model III, we use HoltWinters function in R to compute additive exponential smoothing with trend and level and without seasonal components. We forecast participation in three upcoming time intervals with 80% and 95% confidence. Figure 44 shows the observed and expected contribution patterns.

A manual inspection of Model-II and Model-III shows slight improvement in prediction accuracy of first time interval (refer Figure 43 and Figure 44) of Attrition Rate metric. However, not much difference is observed for second, and third time intervals. We observe similar trends for Regeneration Rate and Retention Rate metrics. In the code snippet that follows, we compare the prediction accuracies of Model-II and Model-III for different horizons (1,2 and 3). We conduct Daibald Mariano Test to compare the predictive accuracies of models (implemented in R language). We hypothesize that the predictive accuracy of linear regression model is less than the predictive accuracy of exponential smoothing model. However, the code snippet extracted from R language shows that with forecast horizons 2 and 3 (for Attrition Rate time series data) the two models are comparable in predictive accuracy. However, for forecast horizon 1 exponential smoothing model outperforms linear regression model. We compute the same statistics for Regeneration Rate and Retention Rate metrics and observe same results for the two models.

```
Diebold-Mariano Test
DM = -1.7906,
Forecast horizon = 3,
Loss function power = 2,
p-value = 0.05044
alternative hypothesis: less
%%%%%%%%%%%
DM = -1.9919,
Forecast horizon = 2,
Loss function power = 2,
p-value = 0.0359
alternative hypothesis: less
%%%%%%%%%%%
DM = -2.8443,
Forecast horizon = 1,
Loss function power = 2,
p-value = 0.00798
alternative hypothesis: less
```

5.5.5 *Discussion*

In FLOSS projects contributor participation follows Pareto Distribution that is 20% of contributors do 80% of work. Also, not all roles are equally relevant. Thus one may argue that it is the contribution pattern of core participants and not all contributors that matters for the project. However, success of FLOSS projects is driven by the masses and not individuals where each contributor has a unique role to serve. For instance, approximately 70% of contributors in FLOSS projects are one time contributors. However, their presence ensures popularity and interest in the project, and is appreciated. So if contributors cease to file issues it is an indicator of decreasing popularity and influences the lifetime of the project. In this study absence of activities for three months implies that contributors left the project. This assumption is localized. For instance, an owner who stop participating for three months indicates that the contributor has left the project while a reporter may continue participation even after a year. However, this assumption indicates the trend of participation. The data available in ITS may not answer all RQs that are relevant to decision makers. However, it gives a justifiable understanding of the stability of the project in a data-driven and objective manner.

5.6 threats to validity

1. *Survey sample size*: We surveyed two very experienced software maintenance professionals in industry. The two professionals worked in very different organizations and presented a range of experience. However, the small sample size is a potential source of bias. Future studies may try to gain perspectives from more such professionals.

2. *Adversary Behavior*: At the time of analysis, the data analyzed in this study is not affected by the adversary behavior. However, in future the choice of metrics may influence the behavior and hence the metrics must be chosen appropriately.

3. *Choice of metrics*: We present 11 metrics for the four roles of software maintenance professionals. This list is not exhaustive, yet covers various aspects of performance. We introduce a framework which can be extended by adding more metrics based on the needs in industry.

4. *Limit on visual points*: In Table 33 we see that the 6 visualization techniques analyze a limited set of developers (count of developers calculated approximately). Beyond this limit, visualization seems cluttered and fail to present the intuition for the plot.

5. *Fixed data dimensionality:* The number of data dimensions that can be analyzed with a visualization technique are fixed. It cannot analyze complex relationships in a higher dimensional dataset (beyond the limit specified for each visualization in Table 33).

6. *Missing data*: We have no way to measure activities that are not recorded in ITS, or activities that are performed between two reported activities in ITS. Also, since we do not consider multiple software repositories that maintains a project we may miss some participation activities for analysis. So the framework present in this study may not measure participation completely. However, assuming that the missing records are uniformly distributed throughout the project, this study provides a fair estimation of the trends which otherwise go unnoticed.

7. *Generalizability*: We conducted experiments on one project. The applicability and usability of the stated approach must be evaluate on various projects.

5.7 summary

In this chapter, we proposed measures the productivity of software contributors by mining software repositories. We found gaps between the perceived relevance of key performance indicators and their measurements in practice. We proposed a framework of metrics for the various roles of software maintenance professionals. Next, we proposed visualizations to presents a panoramic view of developer's contribution in a team and analyzed performance with the perspectives of the managers. These results are validated by survey responses from practitioners in industry. We also present a generalized framework to characterize the stability of software maintenance projects based on community participation patterns. We proposed metrics for three key stability indicators and investigated temporal trends to analyze project stability. We modeled participation pattern to predict future participation.

6 CONCLUSIONS AND FUTURE WORK

When you can measure what you are speaking about, and express it in numbers, you know something about it.

— Lord Kelvin

Software development is a people-intensive activity where contributors create, modify and maintain the software. Much of the earlier works to improve software productivity focused on improvements in process and tools/technology - technical aspects. Relatively less focus was given to people aspects. Recently, the researches on people aspects gained momentum.

This thesis studies contributor productivity to help utilize the capabilities of contributors and improve contributor productivity. It uses Empirical Software Engineering approaches based on Mining Software Repositories to present a quantitative and qualitative analysis of the software development activities. This thesis, through a series of empirical studies, provides frameworks to measure individual and team contributions and explains the influence of various factors on contributor productivity. These studies are conducted on diverse and representative commercial and open source software projects: the popular products at Microsoft, a wide range of software development projects at GitHub and Google Chromium project, to establish the usefulness of the proposed frameworks.

This thesis helps formulate a true picture of software development and facilitates informed decision making. The results of this thesis cross-examines the prevalent intuitions, and backs the intuitions with data. It is to be noted that the results presented in this thesis may not generalize to arbitrary projects. While the results are not generalizable, the frameworks proposed in this thesis can be applied in diverse contexts.

This chapter revisits our studies on contributor productivity and discusses our main contributions. Then we sketch some broader and detailed directions for future research. Additional details are available in the chapter that covers the study.

Factors Influencing the Ramp-up Journey of New Hires

To understand the aids and impediments in the ramp-up journey of newly hired software developers, we conducted a large-scale study. Our analysis combined software engineering data extracted from version control systems and qualitative data from surveys and interviews. We analyzed eight large and popular product groups in Microsoft with several thousand software developers - the majority of engineering workforce at Microsoft. We observed that 14-49% of all software developers in product teams considered in our study are new hires.

This study makes two major contributions. First, it provides a framework to measure the time taken by new hires to reach the productivity level of existing employees. Second, it presents a comprehensive list of factors influencing the ramp-up journey of new hires. This study empirically validates some of the known findings in industry from a systematic analysis of software engineering data.

We measure the ramp-up on two aspects: time to first check-in and time to ramp-up. Time to first check-in marks the time required to make first useful contribution. We measure time to ramp-up as the time required by new hires to reach the median productivity level of the existing employees in the product. Particularly, we measure the time required by new hires to match the frequency of check-ins, lines changed and files changed by existing employees as ramp-up time. These three measures indicate the familiarity with the process, efforts and span of knowledge acquired by new hires respectively.

The results of this study showed that the ramp-up journey is influenced by various factors in the company. We discovered that having a mentor, prior knowledge of required skill sets, etc. helps increase the productivity of new hires. Lack of proper documentation, trying to get access and permissions, etc. reduces the productivity of new hires. Product group of new hires has no influence on the time to ramp-up while career stage of new hires influence it. Various strategic practices like distributed development and prior internship experience within the company are also seen to influence the time to ramp-up. Surprisingly, early first check-in does not imply a faster ramp-up - as expected by the managers.

We found that new hires engage in a wide range of activities other than code check-ins that are not measured in the existing framework. For instance, new hires participate in code reviews, preparing prototypes, reading legacy code, etc. Since, these activities claim time and efforts of new hires and are not measured in the existing framework, care must be taken in evaluating new hires based on these metrics.

New hires also gave suggestions to improve the ramp-up journey based on their experiences. New hires advocated the application of company-wide coding standards, improved code base, improved documentations, easy tools, etc.

Presence of Geographical Bias in Peer Code Review Process

This study analyzes the perceptions and reality of the influence of geographical location on the evaluation of pull requests in GitHub. This study intends to generate awareness and bridge gaps in perceptions, if any. We used a a mixed-methods approach to detect the presence of bias when developers are not aware or not willing to accept their biases. We combined observations from 70,000+ pull requests and 2,500+ survey responses - one of the largest response population of open source projects, to analyse the influence of geographical location on pull request acceptance decisions.

We controlled for the various factors that influence pull request acceptance decisions and quantitatively analysed the GitHub projects' data to measure the influence of 1) geographical location of submitters on pull request acceptance decisions and 2) same geographical location of submitters and integrators on pull request acceptance decisions. We examined pull request acceptance rate across geographical locations as a proxy to geographical bias. We supported these observations with the quantitative and qualitative analysis of the survey responses of submitters and integrators on the perceptions of bias.

The data analysis suggests the presence of geographical bias as there were significant differences in pull request acceptance rates based on geographical locations. This observation was in agreement with the experiences of pull request submitters. We found that the perceptions of submitters classified by geographical locations match the data analysis. However, integrators did not perceive being biased in evaluating submitters. Integrators felt that factors relating to the geographical location of submitters and not necessarily the geographical location may influence their pull request acceptance decisions. Integrators perceived that the observed differences can be explained in terms of language barriers and the inability to communicate.

We cross-compared the observations of the survey and data analysis to explain the differences in pull request acceptance rate as geographical bias or factors related to the geographical location. We found the following: First, there are geographical locations in the list which have higher chances of pull request acceptance and where use of English is not widespread, like Japan. Second, we saw that Switzerland and Germany which are pretty similar in terms of language usage are different in terms of pull request acceptance rate. Third, we sent the survey in different languages like French and received numerous responses stating that GitHub contributors are familiar with basic English and there is no need to translate the survey in local languages. Finally, the large number of survey responses received ensures that our results are not biased by a handful of people who have poor communication skills. Together these observations suggest that language barrier and inability to communicate cannot explain the observed behavior implying the presence of geographical bias. It also indicates a bias blind spot as developers see the impact of bias on the judgement of others while failing to see the impact of bias on their own judgement. This study informs both integrators and submitters about the actual presence of bias despite their reported perceptions.

Rise and Fall of Developer Participation due to Competing Projects

This study analyzes the rise and fall of developer participation in a project when competing projects emerge from its source code - termed as forking, claiming existing and potential developer participation. To investigate the influence of forking on the sustainability of original project, we proposed a quantitative framework to identify projects with forks, characterize projects based on developer participation, and measure changes in developer community participation in the original project after forking. We measured changes in the developer community participation after forking and modeled the influence as 1) decrease, 2) increase, and 3) no effect on the sustainability of the developer community participation in the original project. We modeled the observed behavior in terms of project characteristics to understand its reasons.

A large-scale study of projects hosted on GitHub showed that forking is significant and is going to be more relevant in the future. An analysis of more than 2,000 open source software projects suggest that forking significantly alters the sustainability of the developer community participation in the original project. We found that 18%, 29%, and 53% of the original projects observe *decrease*, *increase*, and *no effect* respectively in the developer community participation after forking. Thus a small, yet significant fraction of projects observed a decline in the sustainability of the developer community participation after forking. This effect is more pronounced in projects ported to GitHub from external sources than projects created in GitHub.

We found that the observed behavior can be explained in terms of the characteristics of the competing projects at the time of forking. We analyzed projects of various sizes: small, medium and large and observed the following. We observed that in small projects increase in the maturity of a project by a year increases the chances of a new project emerging from it by 60%. In medium-size projects, increase in contributors' count by a unit reduces the chances of new project emerging from it by 110%. Increase in the influence of the owner of the original project decreases the chances of starting a new project by 10% in large projects. And increase in the influence of the contributor intending to start the new project increases the chances of starting a new project by 19% in large projects. We also found that domain of the project influences the chances of observing a new project and that more popular the original project, less likely it is that a new project will start from it.

We also found that the maturity of the original project, domain, size of the community of the original and new project, influence of the owner of the original and new project, and popularity of the original and new project influences the rise and fall of developer participation in the original project. For instance, an increase in the influence of the owner of the original project and its popularity decreases the chances of decline in the developer community participation in the original project. This observation explains why fork LibreOffice had no effect on the sustainability of the developer community participation in OpenOffice. Interestingly, some factors have dual effects on the project. For instance, in large projects an increase in the maturity of the original project increases the chances of observing a new project while also not influencing the sustainability of the original project.

Effect of Personality Traits on Levels of Contribution

This study explores the inferential power of the personality traits in explaining the behavior of contributors in various contexts of software development in GitHub. This understanding of the behavior of contributors is important to comprehend the intricacies of team. To do so, we measured the personality profiles of contributors, as evident from their language use in discussions on software development. We used the Big Five model from psychology that measures personality profiles in terms of Openness to Experience, Conscientiousness, Extraversion, Agreeableness and Neuroticism.

We empirically analyzed the relationships of the personality traits of contributors with contributions and context of software development in GitHub. Specifically, we studied differences in the personality traits of sub-communities of contributors with different levels of contributions and project membership status. Also, we studied differences in the personality traits of individuals when they contributed in different contexts. Particularly, we looked for changes in the personality traits with time, type of contribution, role and project membership.

We started the study by proposing that the personality traits of contributors mined from software repositories could offer a way to explain their contributions in different contexts of software development. We were able to present several results that supported our ideas.

We showed that the personality traits of contributors with different levels of contributions are different. We found a steady increase in the Openness to Experience, the Conscientiousness, and the Extraversion as contribution increases. This was expected as contributors with higher levels of contributions are the ones who take lead to ensure project success. These contributors - characterized by being more insightful, more goal-oriented and social, seems intuitive. Further, we found that contributors with higher levels of contributions are lower on the Agreeableness. This is again expected from them to ensure the health of the project.

We found that the project members are more Conscientious, more Extrovert and less Agreeable compared to the non-project members. This is expected from project members that they are rigorous in their work, social to attract participation, and less agreeable to maintain quality software development.

We showed that the personality profiles of most active contributors evolve with time. We found that the most active contributors evolve as more Conscientious, more Extrovert and less Agreeable with time. These characteristics explain at the first place why and how these people became the most active contributors.

We showed that contributors portray different personality traits in different contexts of software development. We found that contributors when working on pull requests are most Conscientious, most Extrovert, most Neurotic and least Agreeable. We believe that this behavior indicates gatekeeping to the entry of quality work and that pull requests mark the highest barrier to entry. Among commits and issues, contributors are most Conscientious, most Neurotic and least Agreeable when working on commits. This implies that after pull requests, commits impose largest barriers to entry to ensure quality software development. It is important to note that contributors are least Extrovert when working on commit. One possible explanation for this is that project members collaborate on commits, unlike issues which are largely discussed by prospective project participants or end users. Finally, while working on issues, contributors are least Conscientious, least Neurotic and most Agreeable. This behavior shows how active contributors facilitate the free inflow of suggestions from the masses.

We found that contributors are more Extrovert when they are member of projects or are reviewing others work. This explains the expectations from the project members and the reviewers of successful projects to maintain cordial terms to attract participation. Other than this, we found that as reporter, contributors are more Open to Experience, more Conscientious, and more Neurotic compared to when they are reviewing others work. Also, we found that when contributors are not the member of the project they more Conscientious, more Agreeable and more Neurotic.

Impact of Role Reputation and Contribution on Developer Participation

This study examines contributor characteristics namely role of participation and amount of work done as a measure of predicting developer participation in Google Chromium project. We presented an approach that classifies contributors based on contribution into three mutually exclusive sets namely non-core team, loose core team and tight core team. We defined attrition as a function of participation in two consecutive time intervals and studied attrition rate for reporter, owner, commenter and cc'ed-contributor and the three classes of contributors.

We observed that the attrition rate of all contributors fluctuates from 27% to 47%. We observed marked differences in the attrition rates for the four roles. We found minimum attrition rate for owner and maximum for reporter. This follows the intuition because majority of reporters are end users and not necessarily contributors to the project. We found that the attrition rate for loose core team and tight core team ranges from 3% to 10% which is less than the attrition rate for non-core team (ranges between 27% and 43%). This indicates that retention in project is directly related to the degree of involvement in the project. Interestingly after initial fluctuations, attrition rate of tight core team is higher than the attrition rate of loose core team indicating that tight core team contributes relatively large however sporadically.

Measuring Individual Contribution

This study proposes measures of individual contributions based on pre-defined objectives, roles, and key-performance indicators. We started with a survey of experienced industry professionals to understand the relevance of performance indicators and their measurements in practice. We found that there are differences in the perceived importance of key performance indicators and their measurements. We proposed 11 metrics for bug reporters, bug triagers, bug owners and collaborators based on key performance indicators. Next, we proposed visualizations to present a holistic view of individual contribution in a team. We validated the proposed framework with 10 professionals in industry. Survey responses suggested that the usefulness of the six visualizations varied considerably. However, general consensus validated their applicability.

Measuring Team Contribution Patterns

This study investigates the stability of community in software maintenance projects by analyzing team contributions pattern. We identified three key stability indicators namely attrition, regeneration, and retention and proposed metrics to measure them. We probed temporal community contribution patterns to answer representative research questions aimed at investigating the inferential ability of the metrics in understanding the stability of the project. We modelled the community participation pattern to estimate future participation.

There are many possible ways in which the work described in this thesis can be extended. Some of the broad research directions and specific problems for the research studies are:

Broad Directions

1. *Empirical Analysis* Various studies in management science, social science and psychology study people aspects. Recent years witnessed a rise in studies that empirically analyzes contributor productivity. More such studies are required to completely explain contributor productivity.

2. *Holistic View* A majority of studies analyze the influence of the factor of interest on contributor productivity. However, a holistic understanding of the context and characteristics that completely explains the behavior is missing.

3. *Real-time or Near Real-time Solutions* The solutions proposed in this thesis and various other existing studies are build on historical data and explain the observed behavior. There is a need to apply the knowledge acquired from such studies in real-time or near real-time. For instance, we need a model that comments on the project stability by analyzing the contemporary characteristics of the project and learnings from similar projects in the past.

4. *Replication* This thesis analyzes representative commercial and open source projects. Further studies are required that replicate the frameworks and help generalize the findings.

Specific Problems

1. *Ramp-up Journey* It will be interesting to study the influence of cultural and work hour differences in different nations on the ramp-up journey of new hires. Also, a study on the ramp-up journey of internal transfers can help understand the usefulness of this practice in large companies.

2. *Personality Traits* We characterized the personality traits of most active projects and contributors. Other studies may explore what differentiates most active projects and contributors from less active projects and contributors. Further studies must be conducted on a wider range of open source projects and commercial projects to establish the baseline.

3. *Measuring Contributor Productivity* Future studies may build interactive tools, combine plots to study complex concepts, compare teams and projects for planning and study the evolution of project and organization.

Part I

APPENDIX

A COURSE WORK, FELLOWSHIP AND INTERNSHIP

A.1 COURSE WORK

CGPA: 8.00
Total course credits: 36
Total PhD thesis credits: 84

Course	Grade	Credits
Data Mining	B	4
Machine Learning	B-	4
Pattern Recognition	B-	4
Randomized Algorithm	B	4
Information Retrieval	B	4
Software Debugging	S	2
Social Network Analysis	S	2
Data Analysis	S	2
Software Testing	S	2
Independent Study (IS) at TRDDC, Pune	A-	4
Empirical Software Engineering IS	A-	2
Mining Software Repositories IS	A-	2

A.2 FELLOWSHIP

I am a recipient of the prestigious TCS PhD Research Fellowship from January 2012 to December 2016.

A.3 INTERNSHIP

I interned at Microsoft Research, Redmond from June to September 2014. During my internship, I studied the aids and impediments in the ramp-up journey of new hires. I worked with Suresh Thummalapenta, Thomas Zimmerman, Nachiappan Nagappan, and Jacek Czerwonka.

BIBLIOGRAPHY

[1] Pankaj Jalote. *CMM in practice: processes for executing software projects at Infosys*. Addison-Wesley Professional, 2000.

[2] Damodaram Kamma and Pankaj Jalote. "Effect of task processes on programmer productivity in model-based testing." In: *Proceedings of the 6th India Software Engineering Conference*. ACM. 2013, pp. 23–28.

[3] Robert B Grady. *Successful software process improvement*. Prentice-Hall, Inc., 1997.

[4] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. "A large scale study of programming languages and code quality in github." In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2014, pp. 155–165.

[5] Frederick P Brooks. *The mythical man-month*. Vol. 1995. Addison-Wesley Reading, MA, 1975.

[6] Gerald M Weinberg. *The psychology of computer programming*. Vol. 932633420. Van Nostrand Reinhold New York, 1971.

[7] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*. Vol. 93. Springer, 2008.

[8] Naresh Kumar Nagwani and Shrish Verma. "Rank-Me: A Java Tool for Ranking Team Members in Software Bug Repositories." In: *Journal of Software Engineering & Applications* 5.4 (2012).

[9] Eric Gilbert and Karrie Karahalios. "CodeSaw: A social visualization of distributed software development." In: *Human-Computer Interaction–INTERACT 2007*. Springer, 2007, pp. 303–316.

[10] Jacob T Biehl, Mary Czerwinski, Greg Smith, and George G Robertson. "FASTDash: a visual dashboard for fostering awareness in software teams." In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2007, pp. 1313–1322.

[11] Minghui Zhou and Audris Mockus. "Does the initial environment impact the future of developers?" In: *Proceedings of the 33rd International Conference on Software Engineering*. ACM. 2011, pp. 271–280.

[12] Jilin Chen, Yuqing Ren, and John Riedl. "The effects of diversity on group productivity and member withdrawal in online volunteer groups." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 821–830.

[13] Ahmed E Hassan. "The road ahead for mining software repositories." In: *Frontiers of Software Maintenance, 2008. FoSM 2008.* IEEE. 2008, pp. 48–57.

[14] Natalia Juristo and Ana M Moreno. *Basics of software engineering experimentation*. Springer Science & Business Media, 2013.

[15] Dag IK Sjoberg, Tore Dyba, and Magne Jorgensen. "The future of empirical methods in software engineering research." In: *2007 Future of Software Engineering*. IEEE Computer Society. 2007, pp. 358–378.

[16] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. "Selecting empirical methods for software engineering research." In: *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311.

[17] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. "Empirical studies of software engineering: a roadmap." In: *Proceedings of the conference on The future of Software engineering*. ACM. 2000, pp. 345–355.

[18] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*. Vol. 351. Houghton Mifflin Boston, 1979.

[19] John W Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[20] F Fabbrini, M Fusani, S Gnesi, and G Lami. "Quality evaluation of software requirement specifications." In: *Proceedings of the Software and Internet Quality Week 2000 Conference*. 2000, pp. 1–18.

[21] Colin Potts and Glenn Bruns. “Recording the reasons for design decisions.” In: *Proceedings of the 10th international conference on Software engineering*. IEEE Computer Society Press. 1988, pp. 418–427.

[22] Paul Hamill. *Unit Test Frameworks: Tools for High-Quality Software Development*. " O’Reilly Media, Inc.", 2004.

[23] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. “How can i improve my app? classifying user reviews for software maintenance and evolution.” In: *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE. 2015, pp. 281–290.

[24] Vladimir Rubin, Christian W Günther, Wil MP Van Der Aalst, Ekkart Kindler, Boudewijn F Van Dongen, and Wilhelm Schäfer. “Process mining framework for software processes.” In: *International Conference on Software Process*. Springer. 2007, pp. 169–181.

[25] Lucian Voinea, Alex Telea, and Jarke J Van Wijk. “CVSscan: visualization of code evolution.” In: *Proceedings of the 2005 ACM symposium on Software visualization*. ACM. 2005, pp. 47–56.

[26] Margaret-Anne D Storey, Davor Čubranić, and Daniel M German. “On the use of visualization to support awareness of human activities in software development: a survey and a framework.” In: *Proceedings of the 2005 ACM symposium on Software visualization*. ACM. 2005, pp. 193–202.

[27] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. “Mining eclipse developer contributions via author-topic models.” In: *Mining Software Repositories, 2007. ICSE Workshops MSR’07. Fourth International Workshop on*. IEEE. 2007, pp. 30–30.

[28] Syed Nadeem Ahsan, Muhammad Tanvir Afzal, Safdar Zaman, Christian Guetl, and Franz Wotawa. “Mining effort data from the oss repository of developer’s bug fix activity.” In: *Journal of IT in Asia* 3 (2010), pp. 67–80.

[29] Tanjila Kanij, Robert Merkel, and John Grundy. “Performance assessment metrics for software testers.” In: *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*. IEEE. 2012, pp. 63–65.

[30] Andrea Capiluppi, Alexander Serebrenik, and Ahmmad Youssef. “Developing an h-index for OSS developers.” In: *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE. 2012, pp. 251–254.

[31] Shen Zhang, Yongji Wang, and Junchao Xiao. “Mining individual performance indicators in collaborative development using software repositories.” In: *Software Engineering Conference, 2008. APSEC’08. 15th Asia-Pacific*. IEEE. 2008, pp. 247–254.

[32] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. “Measuring developer contribution from software repository data.” In: *Proceedings of the 2008 international working conference on Mining software repositories*. ACM. 2008, pp. 129–132.

[33] Paulo Fernandes, Afonso Sales, Alan R. Santos, and Thais Webber. “Performance Evaluation of Software Development Teams: a Practical Case Study.” In: *Electron. Notes Theor. Comput. Sci.* 275 (Sept. 2011), pp. 73–92. ISSN: 1571-0661.

[34] T. Kanij, R. Merkel, and J. Grundy. “Performance assessment metrics for software testers.” In: *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*. 2012, pp. 63 –65.

[35] Yared H. Kidane and Peter A. Gloor. “Correlating temporal communication patterns of the Eclipse open source community with performance and creativity.” In: *Comput. Math. Organ. Theory* 13.1 (Mar. 2007), pp. 17–27. ISSN: 1381-298X.

[36] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. “Measuring developer contribution from software repository data.” In: *Proceedings of the 2008 international working conference on Mining software repositories*. MSR ’08. Leipzig, Germany: ACM, 2008, pp. 129–132. ISBN: 978-1-60558-024-1. DOI: 10.1145/1370750.1370781. URL: http://doi.acm.org/10.1145/1370750.1370781.

[37] Naresh Nagwani and Shrish Verma. “Rank-Me: A Java Tool for Ranking Team Members in Software Bug Repositories.” In: *Journal of Software Engineering and Applications* 5.4 (2012), pp. 255–261.

[38] Quinn C Taylor, James E Stevenson, Daniel P Delorey, and Charles D Knutson. "Author entropy: A metric for characterization of software authorship patterns." In: *Third International Workshop on Public Data about Software Development (WoPDaSDÕ08)*. 2008, p. 6.

[39] Jonathan L Krein, Alexander C MacLean, Daniel P Delorey, Charles D Knutson, and Dennis L Eggett. "Language entropy: A metric for characterization of author programming language distribution." In: *4th Workshop on Public Data about Software Development*. 2009.

[40] RI Kilgour, AR Gray, PJ Sallis, and SG MacDonell. "A fuzzy logic approach to computer software source code authorship analysis." In: (1998).

[41] Jason Kealey and Gunter Mussbacher. *StatSVN: Statistics for SVN Repositories Based on the Open Source Project StatCVS*.

[42] Christine A Halverson, Jason B Ellis, Catalina Danis, and Wendy A Kellogg. "Designing task visualizations to support the coordination of work in software development." In: *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. ACM. 2006, pp. 39–48.

[43] Jon Froehlich and Paul Dourish. "Unifying artifacts and activities in a visual tool for distributed software development teams." In: *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society. 2004, pp. 387–396.

[44] Quang Vu Dang, Christian BAC, Olivier BERGER, and Xuan Sang DAO. "Improving community awareness in software forges by semantical aggregation of tools feeds." In: *3nd International Workshop on Public Data about Software Development (WoPDaSD 2008), Milano, Italy, September 2008*. 2008.

[45] SC Eick, Joseph L Steffen, and Eric E Sumner. "Seesoft-a tool for visualizing line oriented software statistics." In: *IEEE Transactions on Software Engineering* 18.11 (1992), pp. 957–968.

[46] Robert DeLine, Mary Czerwinski, Brian Meyers, Gina Venolia, Steven Drucker, and George Robertson. "Code thumbnails: Using spatial memory to navigate source code." In: *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE. 2006, pp. 11–18.

[47] Robert DeLine, Mary Czerwinski, and George Robertson. "Easing program comprehension by sharing navigation data." In: *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. IEEE. 2005, pp. 241–248.

[48] Gregor McEwan and Saul Greenberg. "Supporting social worlds with the community bar." In: *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*. ACM. 2005, pp. 21–30.

[49] Jacob T Biehl and Brian P Bailey. "Improving interfaces for managing applications in multiple-device environments." In: *Proceedings of the working conference on Advanced visual interfaces*. ACM. 2006, pp. 35–42.

[50] Steven Xia, David Sun, Chengzheng Sun, David Chen, and Haifeng Shen. "Leveraging single-user applications for multi-user collaboration: the coword approach." In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM. 2004, pp. 162–171.

[51] Mark Handley and Jon Crowcroft. "Network text editor (NTE): A scalable shared text editor for the Mbone." In: *ACM SIGCOMM Computer Communication Review*. Vol. 27. 4. ACM. 1997, pp. 197–208.

[52] Martin Michlmayr, Gregorio Robles, and Jesus M Gonzalez-Barahona. "Volunteers in large libre software projects: A quantitative analysis over time." In: *Emerging Free and Open Source Software Practices* (2007), pp. 1–24.

[53] Gregorio Robles and Jesus M Gonzalez-Barahona. "Contributor turnover in libre software projects." In: *Open Source Systems*. Springer, 2006, pp. 273–286.

[54] Gregorio Robles, Jesus M Gonzalez-Barahona, and Israel Herraiz. "Evolution of the core team of developers in libre software projects." In: *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE. 2009, pp. 167–170.

[55] Tracy Hall, Sarah Beecham, June Verner, and David Wilson. "The impact of staff turnover on software projects: the importance of understanding what makes software practitioners tick." In: *Proceedings of the 2008 ACM SIGMIS CPR conference on Computer personnel doctoral consortium and research*. ACM. 2008, pp. 30–39.

[56] Pratyush N Sharma, John Hulland, and Sherae Daniel. "Examining Turnover in Open Source Software Projects Using Logistic Hierarchical Linear Modeling Approach." In: *Open Source Systems: Long-Term Sustainability*. Springer, 2012, pp. 331–337.

[57] Andreas Schilling, Sven Laumer, and Tim Weitzel. "Together but apart: how spatial, temporal and cultural distances affect FLOSS developers' project retention." In: *Proceedings of the 2013 annual conference on Computers and people research*. ACM. 2013, pp. 167–172.

[58] Melvin Blumberg and Charles D Pringle. "The missing opportunity in organizational research: Some implications for a theory of work performance." In: *Academy of management Review* 7.4 (1982), pp. 560–569.

[59] Minghui Zhou and Audris Mockus. "What make long term contributors: Willingness and opportunity in oss community." In: *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press. 2012, pp. 518–528.

[60] Alexander Hars and Shaosong Ou. "Working for free? Motivations of participating in open source projects." In: *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE. 2001, 9–pp.

[61] Ronald H Rasch and Henry L Tosi. "Factors affecting software developers' performance: an integrated approach." In: *MIS quarterly* (1992), pp. 395–413.

[62] Jeffrey A Roberts, Il-Horn Hann, and Sandra A Slaughter. "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects." In: *Management science* 52.7 (2006), pp. 984–999.

[63] Andreas Schilling, Sven Laumer, and Tim Weitzel. "Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects." In: *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE. 2012, pp. 3446–3455.

[64] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. "Gender and tenure diversity in GitHub teams." In: *CHI. ACM*. 2015.

[65] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, and Chris Parnin. *Gender bias in open source: Pull request acceptance of women versus men*. Tech. rep. PeerJ PrePrints, 2016.

[66] Yared H Kidane and Peter A Gloor. "Correlating temporal communication patterns of the Eclipse open source community with performance and creativity." In: *Computational and mathematical organization theory* 13.1 (2007), pp. 17–27.

[67] Peter C. Rigby and Ahmed E. Hassan. "What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List." In: *Proceedings of the Fourth International Workshop on Mining Software Repositories*. MSR '07. IEEE Computer Society, 2007.

[68] Carlos Jensen, Scott King, and Victor Kuechler. "Joining Free/Open source software communities: An analysis of newbies' first interactions on project mailing lists." In: *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE. 2011, pp. 1–10.

[69] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. "Ramp-Up Journey of New Hires: Tug of War of Aids and Impediments." In: *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2015, pp. 1–10. DOI: 10.1109/ESEM.2015.7321212.

[70] Ayushi Rastogi, Suresh Thummalapenta, Thomas Zimmermann, Nachiappan Nagappan, and Jacek Czerwonka. "Ramp-up Journey of New Hires: Do strategic practices of software companies influence productivity?" In: *10th Innovations in Software Engineering Conference (ISEC), formerly India Software Engineering Conference*. 2017, pp. 1–4.

[71] Ayushi Rastogi, Nachiappan Nagappan, and Georgios Gousios. "Geographical Bias in GitHub: Perceptions and Reality." In: *Submitting to ICSE 2018*.

[72] Ayushi Rastogi and Nachiappan Nagappan. "Forking and the Sustainability of the Developer Community Participation–An Empirical Investigation on Outcomes and Reasons." In: *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE. 2016, pp. 102–111. DOI: 10.1109/SANER.2016.27.

[73] Ayushi Rastogi and Nachiappan Nagappan. “On the Personality Traits of GitHub Contributors.” In: *27th International Symposium on Software Reliability Engineering, Ottawa, Canada*. 2016.

[74] Ayushi Rastogi and Ashish Sureka. “Open Source Software: Mobile Open Source Technologies: 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Proceedings.” In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. Chap. Does Contributor Characteristics Influence Future Participation? A Case Study on Google Chromium Issue Tracking System, pp. 164–167. ISBN: 978-3-642-55128-4. DOI: 10.1007/978-3-642-55128-4_22. URL: http://dx.doi.org/10.1007/978-3-642-55128-4_22.

[75] Ayushi Rastogi, Arpit Gupta, and Ashish Sureka. “Samiksha: Mining Issue Tracking System for Contribution and Performance Assessment.” In: *Proceedings of the 6th India Software Engineering Conference*. ISEC ’13. New Delhi, India: ACM, 2013, pp. 13–22. ISBN: 978-1-4503-1987-4. DOI: 10.1145/2442754.2442757. URL: http://doi.acm.org/10.1145/2442754.2442757.

[76] Ayushi Rastogi and Ashish Sureka. “SamikshaViz: A Panoramic View to Measure Contribution and Performance of Software Maintenance Professionals by Mining Bug Archives.” In: *Proceedings of the 7th India Software Engineering Conference*. ISEC ’14. Chennai, India: ACM, 2014, 2:1–2:10. ISBN: 978-1-4503-2776-3. DOI: 10.1145/2590748.2590750. URL: http://doi.acm.org/10.1145/2590748.2590750.

[77] Ayushi Rastogi and Ashish Sureka. “What Community Contribution Pattern Says about Stability of Software Project?” In: *21st Asia-Pacific Software Engineering Conference*. Vol. 2. 2014, pp. 31–34. DOI: 10.1109/APSEC.2014.88.

[78] Ayushi Rastogi and Ashish Sureka. “SamikshaUmbra: Contribution and Performance Assessment of Software Maintenance Professionals by Mining Software Repositories.” In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. Vol. 2. 2013, pp. 170–175.

[79] Eric Brechner. “Things they would not teach me of in college: what Microsoft developers learn later.” In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM. 2003, pp. 134–136.

[80] Andrew Begel and Beth Simon. “Struggles of new college graduates in their first software development job.” In: *ACM SIGCSE Bulletin*. Vol. 40. 1. ACM. 2008, pp. 226–230.

[81] Darryl K Taft. “Programming grads meet a skills gap in the real world.” In: *Retrieved September* (2007).

[82] *Rational Team Concert*. http://www-03.ibm.com/software/products/en/rtc/. [Online; accessed 11-March-2015].

[83] *Team Foundation Server*. http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx. [Online; accessed 11-March-2015].

[84] *git–distributed-even-if-your-workflow-isnt*. http://www.git-scm.com/. [Online; accessed 11-March-2015].

[85] *Survey Identifies Greatest Challenges When Starting a New Job*. http://accountemps.rhi.mediaroom.com/new-job-challenges. [Online; accessed 11-March-2015].

[86] Andrew Begel and Beth Simon. “Novice software developers, all over again.” In: *Proceedings of the Fourth international Workshop on Computing Education Research*. ACM. 2008, pp. 3–14.

[87] Barthélémy Dagenais, Harold Ossher, Rachel KE Bellamy, Martin P Robillard, and Jacqueline P De Vries. “Moving into a new software project landscape.” In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM. 2010, pp. 275–284.

[88] Cheri Ostroff and Steve WJ Kozlowski. “The role of mentoring in the information gathering processes of newcomers during early organizational socialization.” In: *Journal of Vocational Behavior* 42.2 (1993), pp. 170–183.

[89] Susan Elliott Sim and Richard C Holt. “The ramp-up problem in software projects: A case study of how software immigrants naturalize.” In: *Software Engineering, 1998. Proceedings of the 1998 International Conference on*. IEEE. 1998, pp. 361–370.

[90] Alex Radermacher and Gursimran Walia. “Gaps between industry expectations and the abilities of graduates.” In: *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM. 2013, pp. 525–530.

[91] Jacek Czerwonka, Nachiappan Nagappan, Wolfram Schulte, and Brendan Murphy. "CODEMINE: Building a software development data analytics platform at Microsoft." In: *Software, IEEE* 30.4 (2013), pp. 64–71.

[92] Gregorio Robles, Stefan Koch, Jesús M GonZÁlEZ-BARAHonA, and Juan Carlos. "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool." In: *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*. IET. 2004, pp. 51–56.

[93] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. "What's a typical commit? A characterization of open source software repositories." In: *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*. IEEE. 2008, pp. 182–191.

[94] Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. "Does distributed development affect software quality?: an empirical case study of Windows Vista." In: *Communications of the ACM* 52.8 (2009), pp. 85–93.

[95] *How to convert your internship into a full-time job offer*. http://articles.economictimes.indiatimes.com/2014-05-23/news/50055346_1_interns-hsbc-india-citi-india. Accessed: 2016-06-02.

[96] Drake Baer. *Psychologists say white men benefit from these unconscious biases*. http://www.businessinsider.in/Psychologists-say-white-men-benefit-from-these-unconscious-biases/articleshow/46374544.cms. 2015.

[97] Joan C. Williams. *The 5 biases pushing women out of stem*. https://hbr.org/2015/03/the-5-biases-pushing-women-out-of-stem. 2015.

[98] Anna Sandberg. "Competing biases: Effects of gender and nationality in sports judging." In: (2015).

[99] Rachel L Johnson, Somnath Saha, Jose J Arbelaez, Mary Catherine Beach, and Lisa A Cooper. "Racial and ethnic differences in patient perceptions of bias and cultural competence in health care." In: *Journal of general internal medicine* 19.2 (2004), pp. 101–110.

[100] Olof Åslund and Oskar Nordströum Skans. "Do anonymous job application procedures level the playing field?" In: *Industrial & labor relations review* 65.1 (2012), pp. 82–107.

[101] Sujin K Horwitz and Irwin B Horwitz. "The effects of team diversity on team outcomes: A meta-analytic review of team demography." In: *Journal of management* 33.6 (2007), pp. 987–1015.

[102] Shane Curcuru. *Open Source Is Way Whiter And Maler Than Proprietary Software*. http://readwrite.com/2013/12/11/open-source-diversity. 2015.

[103] Elizabeth Dwoskin. *Computers are showing their biases and tech firms are concerned*. http://www.wsj.com/articles/computers-are-showing-their-biases-and-tech-firms-are-concerned-1440102894. 2015.

[104] Walt Scacchi. "Free/open source software development: Recent research results and methods." In: *Advances in Computers* 69 (2007), pp. 243–295.

[105] Matt Asay. *3 key elements that define every open source*. http://readwrite.com/2013/12/11/open-source-diversity. 2015.

[106] Jason Tsay, Laura Dabbish, and James Herbsleb. "Influence of social and technical factors for evaluating contribution in GitHub." In: *Proceedings of the 36th international conference on Software engineering*. ACM. 2014, pp. 356–366.

[107] *GitHub*. https://github.com. 2015.

[108] *Bitbucket*. https://bitbucket.org. 2015.

[109] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. "Perceptions of diversity on GitHub: A user survey." In: *CHASE. IEEE* (2015).

[110] Timothy D Wilson and Nancy Brekke. "Mental contamination and mental correction: unwanted influences on judgments and evaluations." In: *Psychological bulletin* 116.1 (1994), p. 117.

[111] Alexander R Green, Dana R Carney, Daniel J Pallin, Long H Ngo, Kristal L Raymond, Lisa I Iezzoni, and Mahzarin R Banaji. "Implicit bias among physicians and its prediction of thrombolysis decisions for black and white patients." In: *Journal of general internal medicine* 22.9 (2007), pp. 1231–1238.

[112] Nathaniel Branden. *BrainyQuote.com. Xplore Inc.* URL: http://www.brainyquote.com/quotes/quotes/n/nathanielb163773.html.

[113] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective." In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE. Austin, Texas, USA, May 2016. DOI: http://dx.doi.org/10.1145/2884781.2884826. URL: /pub/pullreqs-contributors.pdf.

[114] Marianne Bertrand and Sendhil Mullainathan. *Are Emily and Greg more employable than Lakisha and Jamal? A field experiment on labor market discrimination*. Tech. rep. National Bureau of Economic Research, 2003.

[115] Walt Scacchi. "Free/open source software development: recent research results and methods." In: *ADVANCES IN COMPUTERS* 69 (2006), p. 244.

[116] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. "Impression formation in online peer production: activity traces and personal profiles in github." In: *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM. 2013, pp. 117–128.

[117] Georgios Gousios, Martin Pinzger, and Arie van Deursen. "An exploratory study of the pull-based software development model." In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 345–355.

[118] Nora McDonald and Sean Goggins. "Performance and participation in open source software on github." In: *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2013, pp. 139–144.

[119] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. "Open borders? immigration in open source projects." In: *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE. 2007, pp. 6–6.

[120] Georgios Gousios and Andy Zaidman. "A Dataset for Pull-based Development Research." In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 368–371. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597122.

[121] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Klaus Schneider. "Creating a shared understanding of testing culture on a social coding site." In: *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE. 2013, pp. 112–121.

[122] Peter Weißgerber, Daniel Neu, and Stephan Diehl. "Small patches get in!" In: *Proceedings of the 2008 international working conference on Mining software repositories*. ACM. 2008, pp. 67–76.

[123] Ayushi Rastogi. *Data for replication*. https://sites.google.com/site/ayushirastogi1989india/research/geographicalbiasstudy-data. 2015.

[124] Meir Shemla, Bertolt Meyer, Lindred Greer, and Karen A Jehn. "A review of perceived diversity in teams: Does how members perceive their team's composition affect team processes and outcomes?" In: *Journal of Organizational Behavior* (2014).

[125] Donn Erwin Byrne. *The attraction paradigm*. Vol. 11. Academic Pr, 1971.

[126] *GHTorrent*. http://ghtorrent.org/downloads.html. 2015.

[127] Marek Hlavac. *stargazer: Well-Formatted Regression and Summary Statistics Tables*. R package version 5.2. 2015. URL: http://CRAN.R-project.org/package=stargazer.

[128] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: https://www.R-project.org/.

[129] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/-correlation analysis for the behavioral sciences*. Routledge, 2013.

[130] Uwe Flick. *An introduction to qualitative research*. Sage, 2009.

[131] Harald Cramér. *Mathematical methods of statistics*. Vol. 9. Princeton university press, 1999.

[132] Marie Kraska-Miller. *Nonparametric statistics for social and behavioral sciences*. CRC Press, 2013.

[133] Quang H Vuong. “Likelihood ratio tests for model selection and non-nested hypotheses.” In: *Econometrica: Journal of the Econometric Society* (1989), pp. 307–333.

[134] Adrian Furnham. “Response bias, social desirability and dissimulation.” In: *Personality and individual differences* 7.3 (1986), pp. 385–400.

[135] LEE SIGELAMAN. “Question-order effects on presidential popularity.” In: *Public Opinion Quarterly* 45.2 (1981), pp. 199–207.

[136] Emily Pronin, Daniel Y Lin, and Lee Ross. “The bias blind spot: Perceptions of bias in self versus others.” In: *Personality and Social Psychology Bulletin* 28.3 (2002), pp. 369–381.

[137] Jonas Gamalielsson and Björn Lundell. “Long-Term Sustainability of Open Source Software Communities beyond a Fork: A Case Study of LibreOffice.” In: *Open Source Systems: Long-Term Sustainability*. Springer, 2012, pp. 29–47.

[138] Josh Lerner and Jean Tirole. “The open source movement: Key research questions.” In: *European Economic Review* 45.4 (2001), pp. 819–826.

[139] Jonas Gamalielsson and Björn Lundell. “Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved?” In: *Journal of Systems and Software* 89 (2014), pp. 128–145.

[140] Linus Nyman. “Hackers on Forking.” In: *Proceedings of The International Symposium on Open Collaboration*. ACM. 2014, p. 6.

[141] Linus Nyman, Tommi Mikkonen, Juho Lindman, and Martin Fougère. “Forking: the invisible hand of sustainability in open source software.” In: *Proceedings of SOS*. 2011, pp. 1–5.

[142] Karl Fogel. *Producing open source software: How to run a successful free software project*. " O’Reilly Media, Inc.", 2005.

[143] Steve Weber. *The success of open source*. Vol. 368. Cambridge Univ Press, 2004.

[144] Andrew M St Laurent. *Understanding open source and free software licensing*. " O’Reilly Media, Inc.", 2004.

[145] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. “Social coding in GitHub: transparency and collaboration in an open software repository.” In: *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM. 2012, pp. 1277–1286.

[146] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. “Gender and Tenure Diversity in GitHub Teams.” In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. 2015, pp. 3789–3798.

[147] Gregorio Robles and Jesús M González-Barahona. “A comprehensive study of software forks: Dates, reasons and outcomes.” In: *Open Source Systems: Long-Term Sustainability*. Springer, 2012, pp. 1–14.

[148] Linus Nyman and Tommi Mikkonen. “To fork or not to fork: Fork motivations in SourceForge projects.” In: *Open Source Systems: Grounding Research*. Springer, 2011, pp. 259–268.

[149] Robert Viseur. “Forks impacts and motivations in free and open source projects.” In: *International Journal of Advanced Computer Science and Applications (IJACSA)* 3.2 (2012), pp. 117–122.

[150] Yulin Fang and Derrick Neufeld. “Understanding sustained participation in open source software projects.” In: *Journal of Management Information Systems* 25.4 (2009), pp. 9–50.

[151] Eric S Raymond. *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary*. " O’Reilly Media, Inc.", 2001.

[152] Kam Hay Fung, Aybüke Aurum, and David Tang. “Social Forking in Open Source Software: An Empirical Study.” In: *CAiSE Forum*. Citeseer. 2012, pp. 50–57.

[153] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. “The promises and perils of mining GitHub.” In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM. 2014, pp. 92–101.

[154] Georgios Gousios. “The GHTorrent dataset and tool suite.” In: *Proceedings of the 10th Working Conference on MSR*. MSR ’13. San Francisco, CA, USA: IEEE Press, 2013, pp. 233–236. ISBN: 978-1-4673-2936-1. URL: http://dl.acm.org/citation.cfm?id=2487085.2487132.

[155] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. "The promises and perils of mining git." In: *MSR, 2009. 6th IEEE International Working Conference on*. IEEE. 2009, pp. 1–10.

[156] Walter Mischel and Yuichi Shoda. "A cognitive-affective system theory of personality: reconceptualizing situations, dispositions, dynamics, and invariance in personality structure." In: *Psychological review* 102.2 (1995), p. 246.

[157] Sherlock A Licorish and Stephen G MacDonell. "Communication and personality profiles of global software developers." In: *Information and Software Technology* 64 (2015), pp. 113–131.

[158] Peter C Rigby and Ahmed E Hassan. "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list." In: *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society. 2007, p. 23.

[159] Sherlock A Licorish and Stephen G MacDonell. "What Can Developers' Messages Tell Us? A Psycholinguistic Analysis of Jazz Teams' Attitudes and Behavior Patterns." In: *Software Engineering Conference (ASWEC), 2013 22nd Australian*. IEEE. 2013, pp. 107–116.

[160] Sherlock A Licorish and Stephen G MacDonell. "What affects team behavior?: preliminary linguistic analysis of communications in the jazz repository." In: *Proceedings of the 5th International Workshop on Co-operative and Human Aspects of Software Engineering*. IEEE Press. 2012, pp. 83–89.

[161] Sherlock A Licorish and Stephen G MacDonell. "The true role of active communicators: an empirical study of Jazz core developers." In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM. 2013, pp. 228–239.

[162] Yi Wang. "Building the linkage between project managers' personality and success of software projects." In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society. 2009, pp. 410–413.

[163] Blerina Bazelli, Abram Hindle, and Eleni Stroulia. "On the personality traits of stackoverflow users." In: *2013 IEEE International Conference on Software Maintenance*. IEEE. 2013, pp. 460–463.

[164] Chris Sumner, Alison Byers, Rachel Boochever, and Gregory J Park. "Predicting dark triad personality traits from twitter usage and a linguistic analysis of tweets." In: *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*. Vol. 2. IEEE. 2012, pp. 386–393.

[165] James W Pennebaker, Martha E Francis, and Roger J Booth. "Linguistic inquiry and word count: LIWC 2001." In: *Mahway: Lawrence Erlbaum Associates* 71 (2001), p. 2001.

[166] Paul T Costa and Robert R MacCrae. *Revised NEO personality inventory (NEO PI-R) and NEO five-factor inventory (NEO FFI): Professional manual*. Psychological Assessment Resources, 1992.

[167] Lawrence A Pervin and Oliver P John. *Handbook of personality: Theory and research*. Elsevier, 1999.

[168] Robert R McCrae and Paul T Costa. *Personality in adulthood: A five-factor theory perspective*. Guilford Press, 2003.

[169] Isabel Briggs Meyers. *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Consulting Psychologists Press, Incorporated, 1999.

[170] Lewis R Goldberg. "An alternative" description of personality": the big-five factor structure." In: *Journal of personality and social psychology* 59.6 (1990), p. 1216.

[171] Carl Gustav Jung. *Psychological types*. Routledge, 2014.

[172] Sharon McDonald and Helen M Edwards. "Who should test whom?" In: *Communications of the ACM* 50.1 (2007), pp. 66–71.

[173] François Mairesse, Marilyn A Walker, Matthias R Mehl, and Roger K Moore. "Using linguistic cues for the automatic recognition of personality in conversation and text." In: *Journal of artificial intelligence research* (2007), pp. 457–500.

[174] Lewis R Goldberg. "Language and individual differences: The search for universals in personality lexicons." In: *Review of personality and social psychology* 2.1 (1981), pp. 141–165.

[175] Tal Yarkoni. "Personality in 100,000 words: A large-scale analysis of personality and word use among bloggers." In: *Journal of research in personality* 44.3 (2010), pp. 363–373.

[176] James W Pennebaker, Ryan L Boyd, Kayla Jordan, and Kate Blackburn. "The Development and Psychometric Properties of LIWC2015." In: *UT Faculty/Researcher Works* (2015).

[177] James W Pennebaker and Laura A King. "Linguistic styles: language use as an individual difference." In: *Journal of personality and social psychology* 77.6 (1999), p. 1296.

[178] Adam DI Kramer. "The spread of emotion via Facebook." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2012, pp. 767–770.

[179] Gary Hsieh, Jilin Chen, Jalal U Mahmud, and Jeffrey Nichols. "You read what you value: understanding personal values and reading interests." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2014, pp. 983–986.

[180] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. "Towards discovering the role of emotions in stack overflow." In: *Proceedings of the 6th International Workshop on Social Software Engineering*. ACM. 2014, pp. 33–36.

[181] Alberto Bacchelli, Michele Lanza, and Romain Robbes. "Linking e-mails and source code artifacts." In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM. 2010, pp. 375–384.

[182] LM Hough and JW Johnson. "Use and importance of personality variables in work settings." In: *Handbook of psychology* 12 (2013), pp. 211–243.

[183] Audris Mockus, Roy T Fielding, and James D Herbsleb. "Two case studies of open source software development: Apache and Mozilla." In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.3 (2002), pp. 309–346.

[184] R Core Team. *R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. 2013*. 2014.

[185] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. "Mining Valence, Arousal, and Dominance-Possibilities for Detecting Burnout and Productivity?" In: *arXiv preprint arXiv:1603.04287* (2016).

[186] Adam DI Kramer, Jamie E Guillory, and Jeffrey T Hancock. "Experimental evidence of massive-scale emotional contagion through social networks." In: *Proceedings of the National Academy of Sciences* 111.24 (2014), pp. 8788–8790.

[187] Jorge Colazo and Yulin Fang. "Impact of license choice on open source software development activity." In: *Journal of the American Society for Information Science and Technology* 60.5 (2009), pp. 997–1011.

[188] Daniel Izquierdo-Cortazar, Gregorio Robles, Felipe Ortega, and Jesus M Gonzalez-Barahona. "Using software archaeology to measure knowledge loss in software projects due to developer turnover." In: *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*. IEEE. 2009, pp. 1–10.

[189] Yiqing Yu, Alexander Benlian, and Thomas Hess. "An Empirical Study of Volunteer Members' Perceived Turnover in Open Source Software Projects." In: *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE. 2012, pp. 3396–3405.

[190] Vishal Gupta and Sushil Kumar. "Impact of performance appraisal justice on employee engagement: a study of Indian professionals." In: *Employee Relations* 35.1 (2013), pp. 61–78.

[191] Robin Kessler. *Competency-Based Performance Reviews: How to Perform Employee Evaluations the Fortune 500 Way*. Career Press, 2008.

[192] Herwig Kressler. *Motivate and Reward: Performance Appraisal and Incentive Systems for Business Success*. Palgrave Macmillan, 2003.

[193] Carrie Dusterhoff, J Barton Cunningham, and James N MacGregor. "The Effects of Performance Rating, Leader–Member Exchange, Perceived Utility, and Organizational Justice on Performance Appraisal Satisfaction: Applying a Moral Judgment Perspective." In: *Journal of Business Ethics* (2013), pp. 1–9.

[194] Syed Nadeem Ahsan, Muhammad Tanvir Afzal, Safdar Zaman, Christian Guetl, and Franz Wotawa. "Mining Effort Data from the OSS Repository of Developer's Bug Fix Activity." In: *Journal of IT in Asia* 3 (2010), pp. 67–80.

[195] Andrea Capiluppi and Martin Michlmayr. “From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects.” In: *Open Source Development, Adoption and Innovation*. Springer, 2007, pp. 31–44.

[196] Wenjin Wu, Wen Zhang, Ye Yang, and Qing Wang. “Time series analysis for bug number prediction.” In: *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*. IEEE. 2010, pp. 589–596.

[197] Liguo Yu, Srini Ramaswamy, RB Lenin, and VL Narasimhan. “Time series analysis of open-source software projects.” In: *Proceedings of the 47th Annual Southeast Regional Conference*. ACM. 2009, p. 64.

[198] Mathieu Goeminne and Tom Mens. “A framework for analysing and visualising open source software ecosystems.” In: *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. ACM. 2010, pp. 42–47.

[199] Ping Zhang and Andrew B Whinston. “Business Information Visualization for Decision-Making Support-A Research Strategy.” In: (1995).

[200] Bastiaan Schonhage and Anton Eliëns. “Management through vision-a case study towards requirements of BizViz.” In: *Information Visualization, 2000. Proceedings. IEEE International Conference on*. IEEE. 2000, pp. 387–392.

[201] Nicolas Bettenburg, Sascha Just, Adrian Schroted, Cathrin WeiB, Rahul Premraj, and Thomas Zimmermann. “Quality of bug reports in Eclipse.” In: *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*. eclipse ’07. ACM Digital Library, 2007.

[202] Nicolas Bettenburg, Sascha Just, Adrian Schroted, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. “What makes a good bug report?” In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. SIGSOFT’08/FSE-16. ACM Digital Library, 2008.

[203] Philipp Schugerl, Juergen Rilling, and Philippe Charland. “Mining Bug Repositories- A Quality Assessment.” In: *2008 International Conference on Computational Intelligence for Modelling Control and Automation*. IEEE Computer Society, 2008.

[204] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. “"Not my bug!" and other reasons for software bug report reassignments.” In: *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. CSCW’11. ACM digital library, 2011.

[205] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. “Improving bug triage with bug tossing graphs.” 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. ACM digital library, 2009.

[206] Richard A Becker, William S Cleveland, and Ming-Jen Shyu. “The visual design and control of trellis display.” In: *Journal of computational and Graphical Statistics* 5.2 (1996), pp. 123–155.

[207] Paul Murrell. *R graphics*. Chapman and Hall/CRC, 2005.

[208] Brian Johnson. “TreeViz: treemap visualization of hierarchically structured information.” In: *Human factors in computing systems*. ACM. 1992, pp. 369–370.

[209] Brian Johnson and Ben Shneiderman. “Tree-maps: A space-filling approach to the visualization of hierarchical information structures.” In: *Visualization*. IEEE. 1991, pp. 284–291.

[210] Günther Sawitzki and Statlab Heidelberg. *Bertin Matrices: An R Implementation*. 2012.

[211] S Few. “Bullet graph design specification.” In: *Perceptual Edge-White Paper* (2010).

[212] Mike Frank and Ed Vul. *Supplemental Resource: Brain and Cognitive Sciences Statistics and Visualization for Data Analysis and Inference*. 2009.

[213] I Bernard Cohen. “Florence Nightingale.” In: *Scientific American* 250.3 (1984), pp. 128–137.

[214] Paul J Lewi. *Speaking of graphics*. 2006.

[215] Israel Herraiz, Jesus M Gonzalez-Barahona, Gregorio Robles, and Daniel M German. “On the prediction of the evolution of libre software projects.” In: *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE. 2007, pp. 405–414.