Singapore Management University

Institutional Knowledge at Singapore Management University

Dissertations and Theses Collection (Open Access) | Dissertations and Theses

11-2021

Can we make it better? Assessing and improving quality of GitHub repositories

Gede Artha Azriadi PRANA
Singapore Management University, arthaprana.2016@phdis.smu.edu.sg

Follow this and additional works at: https://ink.library.smu.edu.sg/etd_coll

Part of the Databases and Information Systems Commons, and the Software Engineering Commons

Citation

PRANA, Gede Artha Azriadi. Can we make it better? Assessing and improving quality of GitHub repositories. (2021). 1-197.
Available at: https://ink.library.smu.edu.sg/etd_coll/373

This PhD Dissertation is brought to you for free and open access by the Dissertations and Theses at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Dissertations and Theses Collection (Open Access) by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

CAN WE MAKE IT BETTER? ASSESSING AND IMPROVING QUALITY OF GITHUB REPOSITORIES

GEDE ARTHA AZRIADI PRANA

SINGAPORE MANAGEMENT UNIVERSITY

2021

Can We Make It Better? Assessing and Improving Quality of GitHub Repositories

Gede Artha Azriadi Prana

Submitted to School of Computing and Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee:

David Lo (Supervisor/Chair)
Professor
Singapore Management University

Jing Jiang
Associate Professor
Singapore Management University

Lingxiao Jiang
Associate Professor
Singapore Management University

Asankhaya Sharma
Director of Engineering
Veracode Singapore

Singapore Management University
2021

Copyright (2021) Gede Artha Azriadi Prana

I hereby declare that this PhD dissertation is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree in any university previously.

Gede Artha Azriadi Prana

24 November 2021

Abstract

The code hosting platform GitHub has gained immense popularity worldwide in recent years, with over 200 million repositories hosted as of June 2021. Due to its popularity, it has great potential to facilitate widespread improvements across many software projects. Naturally, GitHub has attracted much research attention, and the source code in the various repositories it hosts also provide opportunity to apply techniques and tools developed by software engineering researchers over the years. However, much of existing body of research applicable to GitHub focuses on code quality of the software projects and ways to improve them. Fewer work focus on potential ways to improve quality of GitHub repositories through other aspects, although quality of a software project on GitHub is also affected by factors outside a project's source code, such as documentation, the project's dependencies, and pool of contributors.

The three works that form this dissertation focus on investigating aspects of GitHub repositories beyond the code quality, and identify specific potential improvements that can be applied to improve wide range of GitHub repositories. In the first work, we aim to systematically understand the content of README files in GitHub software projects, and develop a tool that can process them automatically. The work begins with a qualitative study involving 4,226 README file sections from 393 randomly-sampled GitHub repositories, which reveals that many README files contain the "What" and "How" of the software project, but often do not contain the purpose and status of the project. This is followed by a development and evaluation of a multi-label classifier that can predict eight different README content categories with F1 of 0.746. From our subsequent evaluation of the classifier, which involve twenty software professionals, we find that adding labels generated by the classifier to README files ease information discovery.

Our second work focuses on characteristics of vulnerabilities in open-source libraries used by 450 software projects on GitHub that are written in Java,

Python, and Ruby. Using an industrial software composition analysis tool, we scanned every version of the projects after each commit made between November 1, 2017 and October 31, 2018. Our subsequent analyses on the discovered library names, versions, and associated vulnerabilities reveal, among others, that "Denial of Service" and "Information Disclosure" vulnerability types are common. In addition, we also find that most of the vulnerabilities persist throughout the observation period, and that attributes such as project size, project popularity, and experience level of commit authors do not translate to better or worse handling of vulnerabilities in dependent libraries. Based on the findings in the second work, we list a number of implications for library users, library developers, as well as researchers, and provide several concrete recommendations. This includes recommendations to simplify projects' dependency sets, as well as to encourage research into ways to automatically recommend libraries known to be secure to developers.

In our third work, we conduct a multi-region geographical analysis of gender inclusion on GitHub. We use a mixed-methods approach involving a quantitative analysis of commit authors of 21,456 project repositories, followed by a survey that is strategically targeted to developers in various regions worldwide and a qualitative analysis of the survey responses. Among other findings, we discover differences in diversity levels between regions, with Asia and Americas being highest. We also find no strong correlation between gender and geographic diversity of a repository's commit authors. Further, from our survey respondents worldwide, we also identify barriers and motivations to contribute to open-source software. The results of this work provides insights on the current state of gender diversity in open source software and potential ways to improve participation of developers from under-represented regions and gender, and subsequently improve the open-source software community in general. Such potential ways include creation of codes of conduct, proximity-based mentorship schemes, and highlighting of women / regional role models.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Contribution Summary 2

1.3 Structure of The Dissertation 5

2 Related Works **6**

2.1 Categorizing the Content of GitHub README Files 6

2.1.1 Categorizing software development knowledge 6

2.1.2 Classifying software development text 7

2.1.3 Information needs of software developers 9

2.2 Study of Vulnerabilities of Open-Source Dependencies of GitHub Projects 10

2.2.1 Characteristics of Vulnerabilities 10

2.2.2 Relationship between Software Metrics and Vulnerabilities 12

2.2.3 Vulnerable Dependencies 13

2.3 Diversity of contributors of OSS projects 16

2.3.1 Motivation to contribute to open source software. 16

2.3.2 Barriers to participation in open source. 17

2.3.3 Diversity in open source software projects. 18

3 Categorizing the Content of GitHub README Files **19**

3.1 Introduction and Motivation 19

3.2 Background 22

3.3 Research Methodology 25
3.3.1 Research Questions 25
3.3.2 Data Collection 26
3.3.3 Coding schema 28
3.3.4 Manual annotation 31
3.4 The content of GitHub README files 33
3.4.1 Relations between codes 34
3.4.2 Examples 36
3.5 A GitHub README Content Classifier 38
3.5.1 Overall Framework 39
3.5.2 Feature Extraction 40
3.5.2.1 Statistical Features 40
3.5.2.2 Heuristic Features 41
3.5.3 Classifier Learning 42
3.5.4 Validation 43
3.6 Evaluation of the Classifier 43
3.6.1 Evaluation metric 44
3.6.2 Evaluation results 45
3.6.3 Speed 46
3.6.4 Multi-category sections vs. single-category sections . . . 46
3.6.5 Usefulness of statistical vs. heuristic features 46
3.6.6 Usefulness of particular features 47
3.6.7 Perceived usefulness of automatically labeling sections in GitHub README files 49
3.7 Implications 52
3.8 Threats to Validity 54
3.9 Conclusions and Future Work 55

4 Study on Vulnerabilities in Open Source Software Dependencies 58

4.1 Introduction 58
4.2 Overview of Veracode SCA 63
4.3 Dataset & Methodology 67
4.3.1 Dataset Collection 67
4.3.2 Methodology 70
4.4 Empirical Study Results 71
4.4.1 RQ1: What are the common types and prevalence of dependency vulnerabilities in open-source software, and how persistent are they? 71
4.4.1.1 Dependency vulnerability counts 71
4.4.1.2 Most common dependency vulnerability types . 73
4.4.1.3 Distribution of severity scores 74
4.4.1.4 Vulnerable libraries affecting largest number of sampled projects 76
4.4.1.5 Non-CVE dependency vulnerabilities as discovered by Veracode SCA 78
4.4.1.6 Overall persistence of dependency vulnerabilities 79
4.4.1.7 Change of number of dependency vulnerabilities over the period of observation 82
4.4.1.8 Time required to resolve dependency vulnerabilities 84
4.4.2 RQ2: What are the relationships between dependency vulnerabilities in a project's open-source dependencies with the attributes of the project and its commits? . . . 85
4.4.2.1 Project attributes 85
4.4.2.2 Commit attributes 88
4.5 Discussion and Implications 90
4.5.1 Implications for library users 91
4.5.2 Implications for library developers 92

4.5.3 Implications for researchers 92
4.6 Threats to Validity 93
4.6.1 Threats to internal validity 93
4.6.2 Threats to external validity 94
4.7 Conclusions and Future Work 95

5 Understanding Opportunities and Challenges of Geographic Gender-Inclusion in Open Source Software 96
5.1 Introduction 96
5.2 Methodology 99
5.2.1 Identification of Suitable GitHub Repositories 99
5.2.1.1 Initial Set of Repositories 99
5.2.1.2 Location Resolution of Commit Authors 100
5.2.1.3 Gender Resolution of Commit Authors 101
5.2.1.4 Final Selection of Repositories 102
5.2.1.5 Calculating Gender Diversity of Commit Authors 103
5.2.1.6 Examining Correlation between Geographic and Gender Diversity 106
5.2.1.7 Examining Gender Diversity Changes over Time 107
5.2.1.8 Examining Gender Diversity of Older versus Newer Accounts 107
5.2.2 Globally-Distributed Developer Survey 108
5.2.2.1 Protocol 108
5.2.2.2 Participants 109
5.2.2.3 Analysis 110
5.3 Results 112
5.3.1 RQ1: What are the Gender and Geographic Diversity Characteristics of OSS Projects on GitHub? 112
5.3.1.1 Regional Variations 112

5.3.1.2 Correlation between Geographic and Gender Diversity . . . 114
5.3.1.3 Gender Diversity Changes Over Time 114
5.3.1.4 Gender Diversity of Older versus Newer Accounts 116
5.3.2 RQ 2: What Factors Potentially Contribute to The Differences in Geographic- and Gender-based Developer Participation? . . . 117
5.3.2.1 Global Findings . . . 117
5.3.2.2 Gender and Regional Related Motivations and Challenges . . . 121
5.3.2.3 Regional Variation in Motivations and Challenges 122
5.4 Discussion . . . 125
5.4.1 Summary of Findings . . . 125
5.4.2 Opportunities Ahead . . . 126
5.4.2.1 Development of Friendlier Communities 126
5.4.2.2 Mentorship and Highlighting of Role Models . . 128
5.4.2.3 Diversity Promotion via Automated Software Engineering Tools . . . 129
5.5 Threats to Validity . . . 131
5.6 Conclusion and Future Work . . . 133
5.7 Dataset Availability . . . 134

6 Conclusion and Future Work **135**
6.1 Summary of Contribution . . . 135
6.2 Future Directions . . . 137

7 List of Publications **139**

Bibliography **139**

A Literature Tables **173**

B Survey Results: Encouraging Women **175**

B.1 Encouragement through awareness 175

B.2 Creating opportunities 176

B.3 Outside of GitHub 177

B.4 The “I Don’t Care”s 178

List of Figures

3.1 An excerpt from D3's GitHub README file 23
3.2 Number of sections per README file in our sample 28
3.3 The overall framework of our automated GitHub README content classifier. 39
3.4 Top Features for Each Category. Features starting with *heur_* refer to heuristic features while the remaining features refer to statistical features (see Section 4.2). 48
3.5 An excerpt from a GitHub README file with visual labels, original version at `https://github.com/alt-blog/alt-blog.github.io` 50

4.1 Part of result of a Veracode SCA scan 66
4.2 Overview of Veracode SCA workflow 66
4.3 Relationship between sample projects' commit author count and direct dependency count 70
4.4 Relationship between sample projects' commit author count and transitive dependency count 70
4.5 Kaplan-Meier curve of vulnerable and non-vulnerable libraries detected at first commit. 82

5.1 Gender diversity at region level 2 as of 2014. Darker shade indicates higher diversity. 115
5.2 Gender diversity at region level 2 as per latest data. Darker shade indicates higher diversity. 116

5.3 Gender percentage of commit authors by account creation year, 2014-2018. 117

List of Tables

3.1 Number of repositories excluded from the sample 27

3.2 README section coding reference 30

3.3 Distribution of README categories; App: end-user applications; Lib: libraries; Frame: frameworks; Learn: learning resources, UI: user interfaces 33

3.4 Quantity of codes per section 34

3.5 Association rules at section level 35

3.6 Association rules at file level 35

3.7 Results for Different Classifiers 45

3.8 Effectiveness of Our SVM-based Classifier 45

3.9 Contribution of Different Sets of Features 47

3.10 Questions asked in the survey to determine perceived usefulness of automatically generated section labels 51

3.11 Survey results about the perceived usefulness of automatically labeled GitHub README sections 51

4.1 Veracode SCA vulnerability database information for languages used in this work. Note: Distinct vulnerability corresponds to a CVE for publicly-known vulnerabilities, or Veracode SCA artifact ID for non-publicly-known vulnerabilities. 64

4.2 List of vulnerability tags used by Veracode SCA 67

4.3 Statistics of the sampled projects at latest commit in the observation period. 69

4.4 Correlation between commit author count and commit count, as well as between commit author count and dependency counts 69
4.5 Overview of sample projects' vulnerability counts 72
4.6 Per-project vulnerability percentage distribution by dependency type. 'Both' denotes dependencies that are used both directly and transitively. Includes only projects with at least one vulnerability. Percentages of dependency by type ('Dep.') included for comparison. 73
4.7 Summarized count of dependency vulnerability instances at latest commit. Non-CVE vulnerabilities are identified by Veracode SCA artifact ID 74
4.8 Most common dependency vulnerability tags in each language. "CVE" and "non-CVE" indicate publicly-known and non-publicly-known vulnerabilities, respectively. Instance count and percentage denote count and percentage across the sample set of the programming language. Note that one vulnerability may have more than one tag. 75
4.9 Distribution of severity of vulnerability instances. Percentages are of all vulnerability instances in the respective programming language group (both CVE and non-CVE). 76
4.10 Distribution of severity of top vulnerability types. Percentages are of all vulnerability instances in the respective programming language group (both CVE and non-CVE). 77
4.11 Top vulnerable libraries by projects affected. Project count includes projects using any version of the specified library. 78
4.12 Percentage and top tags for non-CVE vulnerabilities 79
4.13 Per-project survival percentages of vulnerabilities present at first commit, grouped by vulnerability risk rating. 80

4.14 Per-project percentages of dependencies in first commit that remains unchanged throughout the observation period. Median and mean commit counts are shown as indicators of sample projects' activity levels. 81
4.15 Per-project percentage of unchanged dependencies for which newer version already existed at latest commit. Percentages are of all unchanged dependencies in the same project. 82
4.16 Per-project percentage of vulnerabilities that persist despite update of associated dependency. Percentages shown are that of all persistent vulnerabilities in the same project. 82
4.17 Vulnerability and dependency count changes during observation period. T denotes T statistic of Wilcoxon signed-rank test. . . . 83
4.18 Time taken to fix vulnerabilities in days 84
4.19 Negative binomial regression results on project attributes. Shaded cells indicate attributes with statistically significant contribution to dependency vulnerability count. 87
4.20 Counts of the three categories of commits for Java, Python, and Ruby samples 90
4.21 Logistic regression results on commit attributes. Shaded cells indicate attributes with statistically significant contribution to dependency vulnerability count. 90

5.1 Result of project repository filtering steps. 103
5.2 Statistics of shortlisted repositories and associated commit authors. 103
5.3 Commit author region and gender in shortlisted repositories, sorted by Region Level 1. 104

5.4 Diversity and counts of contributors other than commit authors by region level 1. Entries are ordered by non-decreasing Blau index value. Blau index of 0.5 indicate maximum diversity (50% men, 50% women) 106
5.5 Distribution of surveyed commit authors at region level 2. . . . 110
5.6 Distribution of surveyed commit authors at region level 1. . . . 110
5.7 Distribution of survey responses based on gender and region. . . 111
5.8 Gender diversity (or Blau) index arranged in non-decreasing order by region (level 1). Blau index of 0.5 indicate maximum diversity (50% men, 50% women). 112
5.9 Gender diversity index values arranged in non-decreasing order by region (level 2). Blau index of 0.5 indicate maximum diversity (50% men, 50% women). 113
5.10 Gender diversity index values by region level 1, computed by associating project with most frequent contributor location. . . . 113
5.11 Spearman's ρ between repositories' gender diversity and geographic diversity. * indicates p-value <0.001 114
5.12 Changes in gender diversity of commit authors between 2014 and latest GHTorrent date - region level 2. N.A. indicates regions for which Blau index cannot be computed since there are no users at the time. 115
5.13 Motivation of developers to participate in open source software projects across regions. Each cell reports the percentage of developers motivated by the following factors. 122
5.14 Reasons to continue participation in open source software projects across regions. Each cell reports the percentage of developers that find the following factors important or not important. . . . 124
5.15 Relevance of shared regional identity and language across geographic regions. 125

A.1 Comparison of research questions, goals, and findings of closely related literature on gender and geographic diversity. 173
A.2 Comparison of research methods, population / initial sample, and participants' data of closely related literature on gender and geographic diversity. 174

Acknowledgement

I would like to express my gratitude to my supervisor, Prof. David Lo, for his guidance and support throughout my Ph.D journey. I highly appreciate his continuous encouragement, as well as his positive and optimistic attitude towards research, all of which has helped me to grow professionally and personally. I would also like to thank all the committee members of my Ph.D dissertation: Assoc. Prof. Jiang Jing, Assoc. Prof. Jiang Lingxiao, and Dr. Asankhaya Sharma, for the time and effort they have allocated to provide valuable feedback to refine this work.

Next, I would also like to thank Singapore Management University for the scholarship, good research environment, and friendly administrative support, all of which helped my PhD journey immensely. I am also thankful to the friends I've made in SMU, particularly lab mates from Software Analytics Research Group, for the collaboration and camaraderie in the past few years.

Last but not least I would also like to express my heartfelt thanks to my parents, Dr. Made Sri Prana and Dr. Titik Kriswidarti Prana, as well as my wife Sherly and my brother Kresna, all of whom have always supported my learning journey, and been a source of encouragement.

Chapter 1

Introduction

In this chapter, we discuss the motivation of the problems addressed by this dissertation. We also provide a summary of works completed, as well as the structure of this dissertation.

1.1 Background and Motivation

GitHub is a code hosting platform for version control and collaboration[1]. Project artifacts on GitHub are hosted in repositories which can have many branches and are contributed to via commits. Issues and pull requests are the primary artifacts through which development work is managed and reviewed. Over the years, GitHub has gained immense popularity worldwide, with over 200 million repositories hosted[2] as of June 2021. The software projects it hosts range from small-scale personal projects to projects by large organizations such as Adobe, Twitter, and Microsoft[3].

While users and developers of software projects hosted on GitHub often focus on the code hosted on the repository, such code is also associated with other elements, such as documentation and dependencies, that also affect overall quality of the project. A project whose documentation is missing various

[1] https://guides.github.com/activities/hello-world/
[2] https://github.com/about
[3] https://github.com/collections/open-source-organizations

key information may cause problems during usage and thus is not likely to be considered of high quality. Similarly, a project which uses libraries known to be vulnerable will not likely be considered as having high quality, even if the portion of code written by the project owners is secure. Beyond this, a GitHub project also exists within a social context, where people other than the repository owner may contribute to the project. Obstacles preventing capable people from contributing, such as biases against developers from certain gender, can also hinder improvements to a project's quality.

In the past few decades there has been much research on code quality of software projects and ways to improve them. Such research cover topics such as defect prediction [216, 189, 184], fault localization [219, 111, 118], code smells [151, 124, 208], and API recommendation [144, 166, 199]. The results of such research are typically applicable to software projects on GitHub as well. However, there has been less research attention on potential quality improvements in documentation, dependency, and social context of GitHub projects. This situation motivates the body of work described here.

1.2 Contribution Summary

In this dissertation, we focus on analyzing several aspects of software project repositories on GitHub, namely their README files, vulnerabilities in the projects' dependencies, and diversity of their contributors. In addition to analyzing the current state of GitHub with respect to those aspects, we also propose a tool and concrete recommendations to improve GitHub projects based on the result of our analyses.

Categorizing the Content of GitHub README Files. README files play an important role in introducing a software project to potential users and contributors, and the quality of a project's README file affect their impression of the project, for example, whether the project is meant to be a "toy"

project, whether it is still actively developed, and whether it is well-managed. Further, quality of a project's README file also affects how easily users and potential contributors can start using and contributing to the project. While a number of studies have been done on artifacts containing software development knowledge (including API documentation [121], development blogs [154, 201], and StackOverflow [142, 202]), there had not been a systematic study on the content of README files on GitHub and how to automatically process them. Our first work addresses this gap. This work comprises two parts. The first part is an investigation into characteristics and distribution of types of content in GitHub README files. The findings of the first part are used in the second part of the work to develop a multi-label, multi-class classifier for GitHub README file section content. This second part also includes experiments conducted to measure the classifier's performance, along with a survey to evaluate how actual developers perceive the usefulness of the classifier's result.

Study of Vulnerabilities of Open-Source Dependencies of GitHub Projects. Currently, software developers frequently make use of open-source libraries to speed up their development process. The practice even extends to using what is termed by Abdelkareem et al. as "trivial" packages [2] instead of writing the equivalent simple functionalities. However, this reliance on open-source libraries, which themselves often depend on other libraries, makes it more difficult for developers to understand the entire dependency network of their software. Worse, while developers may conduct code reviews on their own code, Kula et al. found that developers often do not scrutinize third-party libraries they use [102]. From security perspective, this means developers are less likely to be aware of any security vulnerabilities introduced by those dependencies into their software project. This has resulted in several high-profile security incidents in recent years that are caused by attackers exploiting vulnerabilities in popular libraries[4][5]. Unfortunately, while cyber-security has been

[4]`https://www.wired.com/story/equifax-breach-no-excuse/`
[5]`https://thehackernews.com/2018/01/electron-js-hacking.html/`

an active research area for decades, characteristics of vulnerabilities in dependent libraries did not gain much research attention until recently. In addition, existing works often focus on very specific set of libraries, such as those in Javascript's npm [44, 227], libraries used by popular websites [110], or libraries most commonly used in a particular organization's projects [156]. In Chapter 4, we conduct a broader study by examining open-source dependencies of 450 GitHub projects of various types that are written in three popular languages (Java, Python, and Ruby) to characterize their vulnerabilities and investigate their correlations with project and commit attributes. Among others, we identify vulnerability types that are common across languages. We also discover that most vulnerabilities in dependent libraries are persistent, that the vulnerability counts do not correlate strongly with commit attributes and most of the project attributes.

Diversity of contributors of OSS projects. The issue of gender diversity in open source software is a long-standing, widespread issue. In a study on popular open source software projects, Bosu and Sultana finds that in all projects analyzed, less than 10% of the core developers are women [18]. A number of other studies report similar findings regarding low gender diversity and prevalence of gender bias [212, 197, 100]. As gender diversity improves productivity of software project teams [147], a lack of diversity will hinder a software project's potential to improve. While gender diversity in software engineering has increasingly gained research attention, existing studies typically treat the set of projects being studied uniformly. This is despite the possibility of difference in motivation, participation level of women, and barrier to contribute between different parts of the world. As software engineering field becomes increasingly globalized, we believe this lack of analysis at regional level becomes increasingly important, and we seek to address this gap. Our work presented in this chapter delves into the cross-section of geography and gender diversity of GitHub repositories. The work analyzes gender diversity

of GitHub contributors in different regions, the change in gender diversity in various regions over time, barriers hindering women to contribute, as well as important factors that motivate or hinder participation of potential contributors in different parts of the world. In addition to reporting the diversity levels, changes, motivations, and barriers, we also provide a number of concrete recommendations that project owners and researchers can implement to promote diversity.

1.3 Structure of The Dissertation

In the remainder of this dissertation, we will first review related work that focus on GitHub in Chapter 2. We subsequently present our work on categorization of GitHub README file content in Chapter 3. Afterwards, we present our work on vulnerability analysis of open-source dependencies of GitHub projects in Chapter 4. We present our third work, which focuses on intersection between geographic and gender diversity, in Chapter 5. Finally, we present the summary of contributions as well as our plans for future work in Chapter 6.

Chapter 2

Related Works

2.1 Categorizing the Content of GitHub README Files

Efforts related to our work on categorization of GitHub README file content can be divided into three categories:

1. Research on categories of software development knowledge
2. Research on classifiers of textual content related to software engineering
3. Studies on the information needs of software developers

2.1.1 Categorizing software development knowledge

Knowledge-based approaches have been extensively used in software development for decades [48], and many research efforts have been undertaken since the 1990s to categorize the kinds of knowledge relevant to software developers [54, 77, 139].

More recently, Maalej and Robillard identified 12 types of knowledge contained in API documentation, with functionality and structure being the most prevalent [121]. Because the authors focused on API documentation, the types

of knowledge they identified are more technical than ours (e.g., containing API-specific concepts such as directives), however, there is some overlap with our categorization of GitHub README files (e.g., in categories such as 'References'). Similar taxonomies have been developed by Monperrus et al. [136] and Jeong et al. [85]. Some of the guidelines identified by Jeong et al. apply to our work as well (e.g., "include 'how to use' documentation") whereas other guidelines are specific to the domain of API documentation or to the user interface through which documentation is presented (e.g., "Effective Search"). Documentation in GitHub README files is broader than API documentation, and the documentation format and its presentation is at least partly specified by the GitHub markdown format.

In addition to API documentation, researchers have investigated the categories of knowledge contained in development blogs [150, 153, 154, 201] and on Stack Overflow [9, 142, 202]. However, these formats serve different purposes compared to GitHub README files, and thus lead to different categories of software development knowledge.

2.1.2 Classifying software development text

The work most closely related to ours in terms of classifying the content of software documentation is OntoCat by Kumar and Devanbu [104]. Using Maalej and Robillard's taxonomy of knowledge patterns in API documentation [121], they developed a domain independent technique to extract knowledge types from API reference documentation. Their system, OntoCat, uses nine different features and their semantic and statistical combinations to classify different knowledge types. Testing OntoCat on Python API documentation, the authors showed the effectiveness of their system. As described above, one major difference between work focused on API documentation and work on GitHub README files is that API documentation tends to be more technical. Similar to our work, Kumar and Devanbu also employed keyphrases for the classifica-

tion, among other features. The F1 scores they report are in a similar range to the ones achieved by our classifier: Their weakest performance was for the categories of Non-Info (0.29) and Control Flow (0.31), while their strongest performance was for the categories of Code Examples (0.83) and Functionality and Behaviour (0.77). In our case, the lowest F1 scores were for the categories of 'Other' (0.303) and 'Reference' (0.605) while the highest scores were for 'How' (0.861) and 'Contribution' (0.814).

In other work focusing on automatically classifying the content of software documentation, Treude and Robillard developed a machine learning classifier that determines whether a sentence on Stack Overflow provides insight for a given API type [204]. Similarly, classifying content on Stack Overflow was the target of Campos et al. [24] and de Souza et al.'s work [41]. Following on from Nasehi et al.'s categorization [142], they developed classifiers to identify questions belonging to different categories, such as 'How-to-do-it'. Also using data from Stack Overflow, Correa and Sureka introduced a classifier to predict deleted questions [35].

Researchers have also applied text classification to bug reports and development issues. For example, Chaparro et al. presented an approach to detect the absence of expected behaviour and steps to reproduce in bug descriptions, aiming to improve bug description quality by alerting reporters about missing information at reporting time [29]. Text classification has also been employed with the goal of automated generation of release notes: Moreno et al. developed a system which extracts changes from source code, summarizes them, and integrates them with information from versioning systems and issue trackers to produce release notes [137]. Abebe et al. used machine learning techniques to automatically suggest issues to be included in release notes [4].

Text classification has also been applied to the information captured in other artifacts created by software developers, including change requests [7], development emails [190], code comments [155], requirements specifications [122],

and app reviews [31, 71, 105, 120].

2.1.3 Information needs of software developers

Although there has not been much work on the information needs of software developers around GitHub repositories, there has been work on information needs of software developers in general. Early work focused mostly on program comprehension [53, 88]. Nykaza et al. investigated what learning support programmers need to successfully use a software development kit (SDK) [145], and they catalogued the content that was seen as necessary by their interviewees, including installation instructions and documentation of system requirements. There is some overlap with the codes that emerged from our analysis, but some of Nykaza et al.'s content suggestions are SDK-specific, such as "types of applications that can be developed with the SDK".

Other studies on the information needs of software developers have analyzed newsgroup questions [82], questions in collocated development teams [98, 203], questions during software evolution tasks [185, 186], questions that focus on issues that occur within a project [65], questions that are hard to answer [109], and information needs in software ecosystems [72]. Information needs related to bug reports have also attracted the attention of the research community: Zimmermann et al. [229] conducted a survey to find out what makes a good bug report and revealed an information mismatch between what developers need and what users supply. Davies and Roper investigated what information users actually provide in bug reports, how and when users provide the information, and how this affects the outcome of the bug [40]. They found that sources deemed highly useful by developers and tools such as stack traces and test cases appeared very infrequently.

The goal of Kirk et al.'s study [97] was understanding problems that occur during framework reuse, and they identified four problems: understanding the functionality of framework components, understanding the interactions

between framework components, understanding the mapping from the problem domain to the framework implementation, and understanding the architectural assumptions in the framework design. These problems will arguably apply to frameworks hosted on GitHub, but not necessarily to other GitHub projects. Our categorization is broader by analyzing the content of GitHub README files for any type of software project. Future work might investigate README files that belong to particular kinds of projects.

2.2 Study of Vulnerabilities of Open-Source Dependencies of GitHub Projects

Research related to our work on vulnerabilities of open-source dependencies of GitHub projects can be categorized as follows:

1. Research into characteristics of vulnerabilities
2. Research into relationship between software metrics and vulnerabilities
3. Research into vulnerable dependencies of software projects

2.2.1 Characteristics of Vulnerabilities

Security vulnerabilities of software projects have been a subject of a number of empirical studies. For example, Shahzad et al. [179] performed analysis on a data set of software vulnerabilities from 1988 to 2011, focusing on seven aspects related to their life cycle. Among other findings, they noted that Denial of Service, Buffer Overflow, and remote code execution are the three most exploited forms of vulnerabilities, but SQL injection, cross-site scripting (XSS), and PHP-specific vulnerabilities were also on the rise. Our findings indicate that at the time of the writing, SQL injection, XSS, and Denial of Service also rank highly among common vulnerability types, although with the exception of Denial of Service, this is not universal across languages. Camilo et

al. [22] performed statistical analyses on bugs and vulnerabilities mined over five releases of Chromium project to examine the relationship between the two groups, and discovered that bugs and vulnerabilities are empirically dissimilar. Ozment and Schechter [149] performed a study on code base of OpenBSD operating system and compiled a database of vulnerabilities identified within a 7.5 year period, and discovered, among others, that 62% of vulnerabilities identified during the period are *foundational*, i.e. the vulnerabilities are already present in the source code at the beginning of the study. Our analysis regarding persistence of vulnerabilities in the sample projects' OSS dependencies found similar vulnerability persistence issues across languages. More recently, Zahedi et al. [221] performed a study on security-related issues from a sample of 200 repositories on GitHub, and discovered that most security issues reported are related to identity management and cryptography, and that security issues comprise only about 3% of all reported issues. We found that in case of vulnerability in OSS dependencies, there is variation across languages. For example, while cryptography-related vulnerabilities ranks among top five in Java, it is not so in other languages. In contrast to the above-mentioned works however, our work focuses on vulnerabilities in the sample projects' OSS dependencies instead of vulnerabilities in the sample projects' source code.

Beyond this, there have also been studies that focus on code and programming practice descriptions in StackOverflow posts. For example, Meng et al. [132] conducted an empirical study on 497 StackOverflow posts related to Java security to understand challenges faced by Java developers in attempting to write secure code. They discovered issues that hinder secure coding practices such as complexity of cryptography APIs and Spring security configuration methods, as well as vulnerabilities in code blocks within accepted answers. Rahman et al. [163] studied code blocks contained in 44,966 Python-related answers on StackOverflow, and found 7.1% of them to contain one or more insecure coding practice, with code injection being the most frequent

type of issue. They also found no relation between user reputation and presence of insecure coding practice in the answer provided by the user. While the scope of our work does not include StackOverflow post, we believe all these factors contribute to spread and persistence of vulnerabilities observed in our study. If a language's security features are difficult to use, and example code in that language commonly contain vulnerabilities, library developers may not be aware of the proper way to write secure code in that language.

2.2.2 Relationship between Software Metrics and Vulnerabilities

A number of work investigate the relationship between the presence of vulnerabilities and various software metrics. Many of those works also propose vulnerability prediction models based on software metrics. For example, Zimmerman et al. [228] investigated whether software metrics that are commonly used in defect prediction (such as code churn and complexity) are useful for predicting vulnerabilities in Windows Vista. They found that using such metrics achieves high precision but low recall. Meneely and Williams [130, 131] examined correlations between vulnerabilities in several open-source software projects and various developer activity metrics, such as number of commits made to a file and number of developers who had changed a file. Among other findings, they found that files that have been changed by six or more developers were four times more likely to contain vulnerability compared to files changed by five or fewer developers. Shin and Williams [182] examined potential usage of execution complexity metrics (such as frequency of function calls) collected from common usage pattern, for predicting software components that may contain vulnerability. They compared the performance between static complexity metrics and the combination of both sets of metrics. They found that the effectiveness of prediction of vulnerable code location using execution complexity metrics vary between the software projects analyzed, with good result for one

of the projects (Firefox) but no significant discriminative ability and low recall in the other (Wireshark).

Other than works focusing on relationship between software metrics and vulnerable part of source code, a number of works use software metrics to identify vulnerability-contributing commits. For example, Bosu et al. [17] analyzed code review requests from 10 open source projects to identify characteristics of changes that are more likely to contain vulnerabilities. They found, among others, that larger number of changed lines correspond to higher likelihood of vulnerability, and that new files are less likely to contain vulnerabilities compared to modified files. Another example is Perl et al.'s work [158] in which they performed mapping between CVEs and GitHub commits of 66 open-source projects and subsequently experimented with using combination of software repositories metadata and software metrics to train a classifier for vulnerability-contributing commit identification. They found that the combination enabled significant reduction of false positives by 99% compared to the then state-of-the-art approach, while maintaining level of recall.

Our work differs from the above-mentioned works since our focus is on vulnerabilities in OSS dependencies of software projects, instead of vulnerabilities in the project code itself. In consequence, our metrics differ. We use project-level metrics and metrics related to software dependencies, instead of file-level metrics. Furthermore, our objective is not vulnerability prediction, but rather investigation into characteristics of projects known to contain OSS dependency vulnerabilities.

2.2.3 Vulnerable Dependencies

There has been several works that discuss vulnerable dependencies in the context of library updatability or migrations. Derr et al. [46] conducted a large-scale library updatability analysis on Android applications along with a survey with developers from Google Play, and reported that among the actively-used

libraries with known security vulnerability, 97.8% can actually be updated without changing application code. They found that reasons for not updating dependencies include lack of incentive to update (since existing versions work as intended), concern regarding possible incompatibility and high integration effort, as well as lack of awareness regarding available updates. Zimmerman et al. [227] conducted a study on security risks in the *npm* ecosystem, and found that single points of failure exists within the ecosystem due to the dependency network structure. The main issues include possibility of vulnerability in single libraries to impact large parts of *npm* ecosystem, and possibility of very small number of package maintainers to introduce vulnerability to large part of the ecosystem. Decan et al. [44] studied the evolution of vulnerabilities in *npm* dependency network using 400 security reports from a 6-year period. Among their findings, they reported that dependency constraints prevented more than 40% of package releases with vulnerable dependencies from being fixed automatically by switching to newer version of the dependencies. Kula et al. [102] conducted a study on impact of security advisories on library migration on 4,600 software projects on GitHub and discovered, among others, that many developers of studied systems do not update vulnerable dependencies and are not likely to respond to a security advisory. Our findings related to persistence of vulnerabilities (Findings 7 and 9 from Chapter 4, Section 4.4) confirm their findings regarding prevalence of significant delay in updating vulnerable dependencies.

Related to the usage of vulnerable dependencies, Cadariu et al. [21] investigated the prevalence of usage of dependencies with known security vulnerabilities in 75 proprietary Java projects built with Maven. They found that 54 of the projects use at least 1 (and up to 7) vulnerable libraries. Lauinger et al. [110] analyzed the usage of Javascript libraries by websites in top Alexa domains as well as random sample of .com websites, and found that around 37% of them include at least one library known to contain vulnerability. Paschenko

et al. [156] performed a study on instances of 200 Java libraries that are most often used in SAP software, and found that about 20% of affected dependencies are not actually deployed, and 81% of vulnerable dependencies can be fixed by simply updating the library version. Dashevskyi et. al. [39] identified three different cost models to estimate the amount of security maintenance effort (e.g. vulnerability fixes) required when using open-source components in proprietary software products. They analyzed usage of 166 open-source components in SAP products and found that open-source component size (measured as lines of code) and age are the main factors influencing security maintenance effort.

Our study on vulnerability in open-source dependencies of GitHub projects uses a larger and more diverse dataset compared to the existing works on vulnerable dependency usage. Our dataset comprises software projects with different characteristics (type, language, authors, organization, etc.), which improves generalizability of our findings. In addition, the software composition analysis tool we use includes a database which includes details on vulnerabilities such as type labels, enabling more systematic grouping and analyses of vulnerability by their characteristics. This allows us to derive insights related to the popularity of different vulnerability types, which has not been analyzed in [21, 110, 156]. Further, the database also includes a number of non-CVE vulnerabilities in addition to publicly-known vulnerabilities in CVE list, which improves comprehensiveness of the scan results. In addition, we scan the dependency graphs of the projects' code bases directly to obtain information on its open-source dependencies and associated vulnerabilities. This approach enables higher accuracy compared to reliance on proxies such as text content of reported issues. Finally, the commit-level granularity of our analysis enables the identification of general changes in the one-year observation period as well as the relationship between vulnerabilities and commit attributes.

2.3 Diversity of contributors of OSS projects

Success of open source software projects is attributed to its developers. This inspired a series of studies exploring reasons for open source engagement. These studies include motivations for developer participation [172], barriers to participation [193], and how developers contribute to open source [230]. These studies help understand and optimize the opportunities to retain community participation. It also prepares projects to avoid or mitigate situations that causes contributors to leave projects.

This chapter is inspired by and extends works on motivation and barriers to participation in open source software projects along the lines of diversity in terms of gender and region of contributors in software projects. Next, we present important studies that have shaped this area of research.

2.3.1 Motivation to contribute to open source software.

Motivation in software engineering has been subject to numerous studies, including several systematic reviews [11, 63, 64]. The existing body of works include a number of studies focusing specifically on motivation of OSS contributors. For example, in a 2002 study, Hars and Ou [6] surveyed open source developers and found that their motivation for contributing are diverse – while students and hobbyists tend to be internally motivated, there are also a large number of developers who are motivated by external rewards. Lakhani and Wolf [106] surveyed 684 OSS developers and found that the strongest type of motivation among the respondents are enjoyment-based intrinsic motivation. Von Krogh et al. [215] examined prior literature on OSS developers' motivation to contribute, and proposed 10 clusters of motivation types categorized into intrinsic motivation, internalized extrinsic motivation, and extrinsic motivation. Barcomb et al.[10] surveyed episodic (non-habitual) OSS volunteers and found that intention to remain are positively associated with social norms, satisfac-

tion, and community commitment. Further, they also found some differences based on participants' gender. Most recently, a study by Gerosa et al. [67] investigated how main motivations of OSS contributors as a group change over the years and how OSS contributors' individual motivations change as they become more experienced. They found that among OSS contributors, some motivations related to social aspect have gained popularity in recent years. They also found that experienced OSS contributors tend to be motivated by intrinsic factors such as altruism, unlike new contributors who tend to place higher importance on factors such as career and learning. These studies facilitate better understanding of what drives people to contribute to OSS projects, what approaches project owners can take to attract contributors, and how these contributors can be retained.

2.3.2 Barriers to participation in open source.

A number of studies investigate barriers that can prevent developers from participating to open source. These barriers have been identified in tools, processes [128], and social collaborations [193]. For example, a study by Terrell et al. [197] found that while women have higher overall acceptance rate of pull requests, their acceptance rate is lower than men when their gender are identifiable and they are not insiders to a project. Another study by Rastogi et al. [168], which analyzes pull requests from 17 countries, found that acceptance rate of contributions can vary significantly depending on the contributor's country of origin, and are higher when when they are evaluated by developers from the same country. The study however does not analyze gender as a factor, as they noted that including only pull requests for which gender data can be obtained will result in sample size that is too small. Other studies examine barriers such as those affecting acceptance of contribution from newcomers [193] or those affecting underrepresented communities [61]). These studies not only help in raising awareness of existence of such barriers, but they

also help in identifying the source of problem. Further, studies such as [194] also propose solutions that can be adopted by OSS community to mitigate such barriers.

2.3.3 Diversity in open source software projects.

In line with increasing awareness regarding the importance of diversity in broader work context, diversity in open source software projects has gained increasingly widespread attention. Starting from the awareness of diversity and particularly the demographic attributes of developers [210, 173], today improving diversity is seen as a goal for fairness [197] as well as improved productivity [211]. Many studies relating to gender diversity and the lack thereof followed, discussing its relevance [197], state of diversity among popular OSS projects [18], male and female OSS contributors' perceptions of other contributors [112], perceptions of women core developers in OSS projects [25], and the impediments to improve gender diversity [84].

All these studies identify challenges and needs of underrepresented communities. We conduct a comparison outlining the distinction between our work and closely-related prior work in Tables A.1 and A.2 in Appendix A.

Our study has common elements to the developer survey on Stack Overflow users [230] but their report does not provide empirical data to support the full scope of motivations and how they persist across genders or regions. Our work provides novelty by conducting an analysis of the activity and experiences at the intersection of gender and global geographic region. Taking research on the subject a step further, in this work we study gender diversity in different regions and how factors relating to gender and region can potentially explain why developers join open source software projects, select a project, continue participation. Such factors can potentially also explain barriers and reasons to leave a project.

Chapter 3

Categorizing the Content of GitHub README Files

3.1 Introduction and Motivation

The `README.md` file for a repository on GitHub is often the first project document that a developer will see when they encounter a new project. This first impression is crucial, as Fogel [57] states: "The very first thing a visitor learns about a project is what its home page looks like. [...] This is the first piece of information your project puts out, and the impression it creates will carry over to the rest of the project by association."

With more than 25 million active repositories at the end of 2017[1], GitHub is the most popular version control repository and Internet hosting service for software projects. When setting up a new repository, GitHub prompts its users to initialize the repository with a `README.md` file which by default only contains the name of the repository and is displayed prominently on the homepage of the repository.

A recent blog post by Christiano Betta[2] compares the README files of four popular GitHub repositories and stipulates that these files should (1)

[1] `https://octoverse.github.com/`

[2] `https://betta.io/blog/2017/02/07/developer-experience-github-readmes/`

inform developers about the project, (2) tell developers how to get started, (3) document common scenarios, and (4) provide links to further documentation and support channels. In its official documentation[3], GitHub recommends that a README file should specify "what the project does, why the project is useful, how users can get started with the project, where users can get help with your project, and who maintains and contributes to the project". Brian Doll of GitHub claimed in a recent interview for IEEE Software that "the projects with good README files tend to be the most used, too, which encourages good README writing behavior" [12].

In the research literature, GitHub README files have been used as a source for automatically extracting software build commands [75], developer skills [70, 76], and requirements [159]. Their content has also played a role in cataloguing and finding similar repositories [180, 223] as well as in analyzing package dependency problems [43].

However, up to now and apart from some anecdotal data, little is known about the content of these README files. To address this gap, our first research question RQ1 asks, *What is the content of GitHub README files?* Knowing the answer to this question would still require readers to read an entire file to understand whether it contains the information they are looking for. Therefore, our second research question RQ2 investigates, *How accurately can we automatically classify the content of sections in GitHub README files?*. To understand a README file's most defining features, our third research question RQ3 asks, *What value do different features add to the classifier?*. Finally, to evaluate the usefulness of the classification, our last research questions RQ4 investigates, *Do developers perceive that the automated classification makes it easier to discover relevant information in GitHub README files?*

To answer our research questions, we report on a qualitative study of a statistically representative sample of 393 GitHub README files containing a

[3]`https://help.github.com/articles/about-readmes/`

total of 4,226 sections. Our conclusions regarding the frequency of section types generalize to the population of all GitHub README files with a confidence interval of 4.94 at a confidence level of 95%. Our annotators and ourselves annotated each section with one or more codes from a coding schema that emerged during our initial analysis. This annotation provides the first large-scale empirical data on the content of GitHub README files. We find that information discussing the 'What' and 'How' of a repository is common while information on purpose and status is rare. These findings provide a point of reference for the content of README files that repository owners can use to meet the expectations of their readers as well as to better differentiate their work from others.

In addition to the annotation, we design a classifier and a set of features to predict categories of sections in the README files. This enables both quick labeling of the sections and subsequent discovery of relevant information. We evaluated the classifier's performance on the manually-annotated dataset, and identify the most useful features for distinguishing the different categories of sections. Our evaluation shows that the classifier achieves an F1 score of 0.746. Also, the most useful features are commonly related to some particular words, either due to their frequency or their unique appearance in sections' headings. In our survey to evaluate the usefulness of the classification, the majority of twenty software professionals perceived the automated labeling of sections based on our classifier to ease information discovery in GitHub README files.

We make the following contributions:

- A qualitative study involving the manual annotation of the content of 4,226 sections from 393 randomly selected GitHub README files, establishing a point of reference for the content of a GitHub README file. We distinguish eight categories in the coding schema that emerged from our qualitative analysis (What, Why, How, When, Who, References, Contribution, and Other), and we report their respective frequencies and

associations.

- We design and evaluate a classifier that categorizes README sections, based on the categories discovered in the annotation process.

- We design and conduct a survey to evaluate the usefulness of the classification by (i) using the automatically determined classes to label sections in GitHub README files using badges and (ii) showing files with and without these badges to twenty software professionals.

We describe background materials on GitHub and README files of repositories hosted there in Section 3.2. We describe our manual annotation methodology in Section 3.3 and the results of the annotation in Section 3.3.4. Section 3.5 introduces the classifier we built for sections of GitHub README files, which we evaluate in Section 3.6. We discuss the implications of our work in Section 3.7 and present the threats to validity associated with this work in Section 3.8. We conclude the work in Section 3.9.

3.2 Background

GitHub is a code hosting platform for version control and collaboration.[4] Project artifacts on GitHub are hosted in repositories which can have many branches and are contributed to via commits. Issues and pull requests are the primary artifacts through which development work is managed and reviewed.

Due to GitHub's pricing model which regulates that public projects are always free[5], GitHub has become the largest open source community in the world, hosting projects from hobby developers as well as organizations such as Adobe, Twitter, and Microsoft.[6]

Each repository on GitHub can have a README file to "tell other people why your project is useful, what they can do with your project, and how

[4] https://guides.github.com/activities/hello-world/
[5] https://github.com/open-source
[6] https://github.com/collections/open-source-organizations

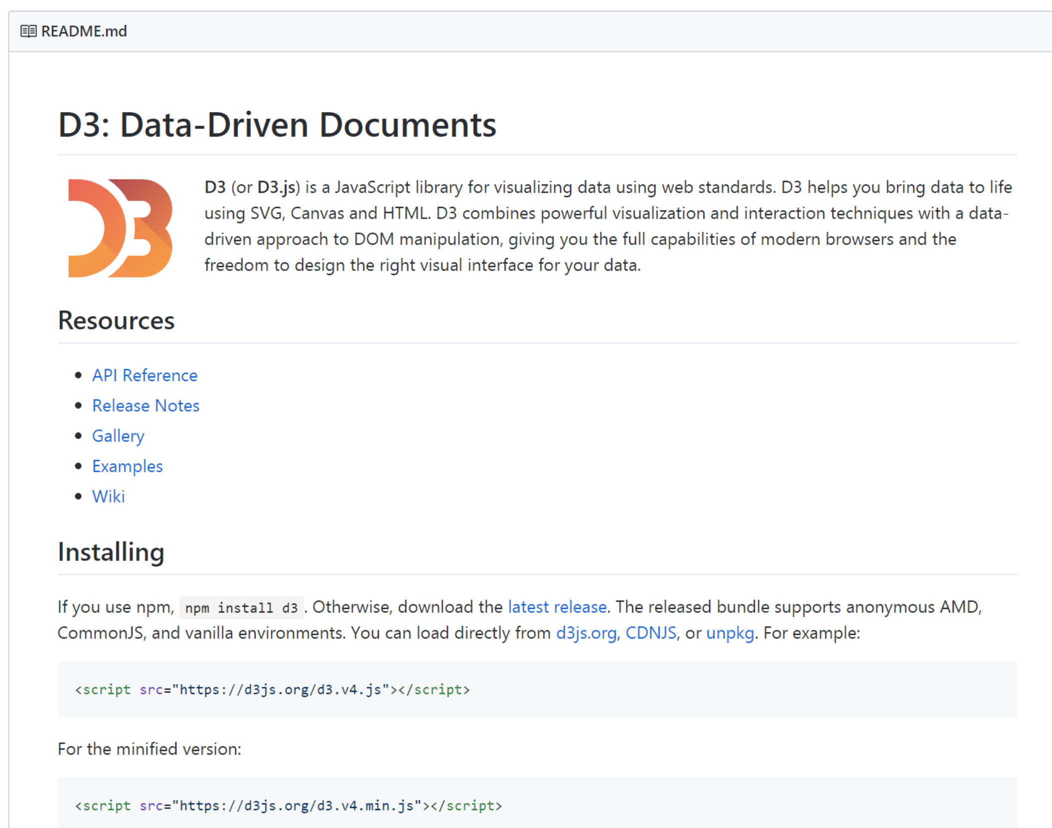

Figure 3.1: An excerpt from D3's GitHub README file

they can use it."[7] README files on GitHub are written in GitHub Flavored Markdown, which offers special formatting for headers, emphasis, lists, images, links, and source code, among others.[8] Figure 3.1 shows the README file of D3, a JavaScript library for visualizing data using web standards.[9] The example shows how headers, pictures, links, and code snippets in markdown files are represented by GitHub.

With 1 billion commits, 12.5 million active issues, and 47 million pull requests in the last 12 months, GitHub plays a major role in today's software development landscape.[10] In 2017, 25 million active repositories were competing for developers' attention, and README files are among the first documents that a developer sees when encountering a new repository.

To gain an understanding of readers' expectations about README files,

[7]https://help.github.com/articles/about-readmes/
[8]https://guides.github.com/features/mastering-markdown/
[9]https://github.com/d3/d3
[10]https://octoverse.github.com/

in our survey to evaluate our classifier, we asked participants what content they expect to find in the README file of a GitHub repository and what single piece of information they would consider most important to be included. Twenty professionals answered our survey—we refer readers to Section 3.6.7 for details on survey design and participant demographics. Here, we summarize the responses we received regarding readers' expectations about the content of GitHub README files.

In response to the open-ended question "What content do you expect to find in the README file of a GitHub repository?", participants mentioned usage instructions (five participants), installation instructions (three participants), prerequisites (three participants), repository license (two participants), purpose of the repository and target audience (two participants), known bugs and trouble-shooting tips (two participants), coding style (one participant), contribution guidelines (one participant), change log (one participant), and screenshots (one participant). For example, one participant answered "Information about the program, how to use it, parameters (if applicable), trouble-shooting tips (if applicable)" and another indicated "I expect to see how to install and run the program successfully". Nine of the twenty participants provided generic answers, such as "More technical information and guidance" and "updates".

In response to "What single piece of information would you consider most important to be included in a GitHub README file?", we also received twenty responses. Usage instructions (e.g., "How to use the features or components of the repository") and license information (e.g., "With my job it's most important to know the licensing information") were identified as most important by three participants each. Two participants indicated known bugs and trouble-shooting tips as being most important, while the other participants mentioned a variety of types of information including target audience, coding style, contribution guidelines, testing information, prerequisites, screenshots and demos,

and project type.

In this work, we study and classify the content of README files on GitHub to investigate the extent to which these expectations are met.

3.3 Research Methodology

In this section, we present our research questions and describe the methods for data collection and analysis.

3.3.1 Research Questions

Our work was guided by four research questions, which focus on categorizing the content of GitHub README files and on evaluating the performance and usefulness of our classifier:

RQ1 What is the content of GitHub README files?

Answers to this question will give insight to repository maintainers and users about what a typical README file looks like. This can serve as a guideline for repository owners who are trying to meet the expectations of their users, and it can also point to areas where owners can make their repositories stand out among other repositories.

RQ2 How accurately can we automatically classify the content of sections in GitHub README files?

Even after knowing what content is typically present in a GitHub README file, readers would still have to read an entire file to understand whether it contains the kind of information they are looking for. An accurate classifier that can automatically classify sections of GitHub README files would render this tedious and time-consuming step unnecessary. From a user perspective, an automated classifier would enable a more structured approach to searching and navigating GitHub README files.

RQ3 What value do different features add to the classifier?

Findings to our third research question will help practitioners and researchers understand the content of README files in more detail and shed light on their defining features. These findings can also be used in future work to further improve the classification.

RQ4 Do developers perceive that the automated classification makes it easier to discover relevant information in GitHub README files?

The goal of our last research question is to evaluate the usefulness of the automated classification of sections in GitHub README files. We use the automatically determined classes to label sections in unseen GitHub README files using badges, and we show GitHub README files with and without these labels to developers and capture their perceptions regarding the ease of discovering relevant information in these files.

3.3.2 Data Collection

To answer our research questions, we downloaded a sample of GitHub `README.md` files[11] by randomly selecting GitHub repositories until we had obtained a statistically representative sample of files that met our selection criteria. We excluded README files that contained very little content and README files from repositories that were not used for software development. We describe the details of this process in the following paragraphs.

To facilitate the random selection, we wrote a script that retrieves a random GitHub repository through the GitHub API using the *repositories* API call in form of `https://api.github.com/repositories?since=<number>`. In this case, `<number>` is the repository ID and was replaced with a random number between 0 and 100,000,000, which was a large enough number to capture

[11] We only consider `README.md` files in our work since these are the ones that GitHub initializes automatically. GitHub also supports further formats such as `README.rst`, but these are much less common and out of scope for this study.

Table 3.1: Number of repositories excluded from the sample

Reason for Exclusion	Repositories
Software, but small README file, i.e., $<$ 2 KB	429
Not software, but large enough README file	127
Not software and small README file	196
README file not in English	48
Number of repositories included in the sample	393
Total number of repositories inspected	1,193

all possible repositories at the time of our data collection. We repeated this process until we had retrieved a sufficient number of repositories so that our final sample after filtering would be statistically representative. We excluded repositories that did not contain a README file in the default location.

Following the advice of Kalliamvakou et al. [90], we further excluded repositories that were not used for software development by inspecting the programming languages automatically detected for each repository by GitHub. If no programming language was detected for a repository, we excluded this repository from our sample.

We manually categorized the README files contained in our samples as end-user applications, frameworks, libraries, learning resources, and projects related to UI. The majority of our README files were related to end-user applications (i.e., 42%) which includes client/server applications, apps/games, plugins, engines, databases, extensions, etc. The second largest category of files was related to libraries (27.9%). Our sample also contained README files related to programming learning resources (17.4%) such as tutorials, assignments, and labs. The remaining files were categorized as frameworks (7.3%) and user interfaces (5.4%) such as CSS styles and images.

We also excluded repositories for which the README file was very small. We considered a file to be very small if it contained less than two kilobytes of data. This threshold was set based on manual inspection of the files which revealed that files with less than two kilobytes of content typically only con-

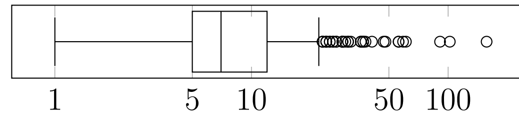

Figure 3.2: Number of sections per README file in our sample

tained the repository name, which is the default content of a new README file on GitHub.

During the manual annotation (see Section 3.3.4), we further excluded README files if their primary language was not English. Table 3.1 shows the number of repositories excluded based on these filters. Our final sample contains 393 README files, which results in a confidence interval of 4.94 at a confidence level of 95% for our conclusions regarding the distribution of section types in the population of all GitHub repositories, assuming a population of 20 million repositories.

We then used GitHub's markdown[12] to extract all sections from the README files in our sample, yielding a total of 4,226 sections distributed over the 393 README files. GitHub's markdown offers headers at different levels (equivalent to HTML's `h1` to `h6` tags) for repository owners to structure their README files. Figure 3.2 shows the distribution of the number of sections per README file. The median value is seven and 50% of the files contain between five and twelve sections.

3.3.3 Coding schema

We adopted 'open coding' since it is a commonly used methodology to identify, describe, or categorize phenomena found in qualitative data [34]. In order to develop a coding scheme, we manually classified a random sample of fifty README files into meaningful categories (known as codes [134]). Our findings from this examination consist of a tentative list of seven categories (e.g. what, why, how) and sub categories (e.g., introduction, background). After defining

[12] `https://guides.github.com/features/mastering-markdown/`

initial codes, we trialed them on 150 README sections using two annotators. For this round of coding, we obtained inter-rater reliability of 76%. Following this trial, we refined our codes until we reached agreement on a scheme that contained codes for all of the types of README sections we encountered. Finally, we define the 'other' category only when all other possibilities have been exhausted. Table 3.2 shows the finalized set of categories as well as example section headings for each category found in this initial sample of README files. The categories roughly correspond to the content of README files that is recommended by GitHub (cf. Introduction).

We identified the first category ('What') based on headings such as 'Introduction' and 'About', or based on the text at the beginning of many README files. We found that either a brief introduction or a detailed introduction is common in our dataset. Conversely, category two ('Why') is rare in README files. For instance, some repositories compare their work to other repositories based on factors such as simplicity, flexibility, and performance. Others list advantages of their project in the introduction.

The most frequent category is 'How' since the majority of README files tend to include instructions on *how to use the project* such as programming-related content (e.g., configuration, installation, dependencies, and errors/bugs). Table 3.2 lists a sample of section headings that belongs to the 'How' category. Further, it is also important to the reader of a README file to be familiar with the status of the project, including versions as well as complete and in-progress functionality. We categorize this kind of time-related information into the fourth code ('When').

We categorize sections as 'Who' content when they include information about who the project gives credit to. This could be the project team or acknowledgements of other projects that are being reused. This category also includes information about licence, contact details, and code of conduct. The second most frequent category is 'References'. This category includes links to

Table 3.2: README section coding reference

#	Category	Example section headings
1	What	Introduction, project background
2	Why	Advantages of the project, comparison with related work
3	How	Getting started/quick start, how to run, installation, how to update, configuration, setup, requirements, dependencies, languages, platforms, demo, downloads, errors and bugs
4	When	Project status, versions, project plans, roadmap
5	Who	Project team, community, mailing list, contact, acknowledgement, licence, code of conduct
6	References	API documentation, getting support, feedback, more information, translations, related projects
7	Contribution	Contributing guidelines
8	Other	

further details such as API documentation, getting support, and translations. This category also includes 'related projects', which is different from the 'comparison with related projects' in category 'Why' due to the lack of an explicit comparison. Our final category is 'Contribution', which includes information about how to fork or clone the repository, as well as details on how to contribute to the project. Our manual analysis indicated that some repositories include separate `CONTRIBUTING.md` files which contain instructions on how to get involved with the project. We do not consider `CONTRIBUTING.md` files in this study. In addition, we included a category called 'Other' which is used for sections that do not belong to any of the aforementioned seven categories.

3.3.4 Manual annotation

We initially used two annotators to code the dataset. One of the annotators was a PhD candidate specializing in Software Engineering while the other one is an experienced Software Engineer working in industry. Each annotator spent approximately thirty hours to annotate the dataset. The task of an annotator is to read the section headings and contents and assign a code based on the coding reference. The annotators assign codes from the eight available codes (Table 3.2). Each section of a README file can have one or more codes.

We measured the inter-rater agreement (i.e. Kappa) between the two annotators and obtained an agreement of 0.858. We used a third annotator to rectify the sections which had no agreement. For this, we annotated the remaining sections that had no agreement. For all cases, we then used a majority vote to determine the final set of codes for each section, i.e., all codes that had been used by at least two annotators for a section were added to the final set of codes for that section.[13] In very few cases, there was still no agreement on any set of codes after considering the codes from three annotators. These cases were manually resolved by discussion.

We manually examined the instances where the annotators disagree. Annotators were likely confused when the README file includes 'Table Of Contents (TOC)' as they have provided inconsistent codes in these instances. Since TOC is included at the beginning of the file, one annotator considers it as category 'What' while the other one placed it in the references. However, the third annotator categorized TOC into 'Other', which is what we used in the final version of the annotated dataset. Another common confusion occurred when categorizing 'community-related' content. Our coding reference (Table 3.2) suggests that community-related information should be placed in the 'Who' category. However, one annotator identified it in the 'Contribution' category. We generally resolved 'community-related' disagreements by placing them into

[13]In cases where there was perfect agreement between the two annotators, the majority vote rule simply yields the codes that both annotators agreed on.

the 'Who' category, in accordance with our coding guide.

We also noticed that our annotators are reluctant to place content into the 'Other' category. Instead, they attempted to classify README contents into the other seven categories. Further, one of the main reasons for disagreement was the inclusion of external links as section titles or contents. For example, one README file listed the middleware available to use with their project as section titles. However, these section titles include "Apache" and "Nginx".[14] One annotator categorized these sections into 'How' while the other placed them in additional resources (code 'References') since they have external links. There can be multiple headings which depend on this decision. For instance, one README file contained 36 headings about configurations. They are categorized into 'How' by one annotator while the other one placed them in additional resources since they have URLs. Resolving this disagreement affected many sections at once.

Further, some README files include screenshots or diagrams to provide an overview or demonstrations. These are expected to be classified in 'Other'. However, annotators have occasionally assigned codes such as 'What', 'How', and 'References' to image contents. Another challenging decision occurs when repositories include all the content under a single heading. This causes the annotators to assign multiple codes which possibly do not overlap between annotators. In addition, we sometimes found misleading headings such as 'how to contribute' where the heading would suggest that the content belongs to category 'Contribution'. However, in a few cases, the content of this section included information on 'how to use the project' (i.e., download, install, and build).

[14] `https://github.com/microlv/prerender`

Table 3.3: Distribution of README categories; App: end-user applications; Lib: libraries; Frame: frameworks; Learn: learning resources, UI: user interfaces

#	Category	# Sections (%)	# Files (%)	App (%)	Lib (%)	Frame (%)	Learn (%)	UI (%)
1	What	707 (16.7%)	381 (97.0%)	14.0	14.2	12.3	22.6	9.6
2	Why	116 (2.7%)	101 (25.7%)	2.6	2.4	3.2	2.6	0.3
3	How	2,467 (58.4%)	348 (88.5%)	49.5	45.0	52.9	52.9	65.6
4	When	180 (4.3%)	84 (21.4%)	5.8	2.5	4.4	0.6	1.3
5	Who	322 (7.6%)	208 (52.9%)	6.6	9.5	5.9	3.7	6.3
6	References	858 (20.3%)	239 (60.8%)	18.4	22.2	17.2	13.5	10.3
7	Contribution	122 (2.9%)	109 (27.8%)	2.4	2.7	3.2	1.6	2.6
8	Other	58 (1.4%)	27 (6.9%)	0.5	1.4	0.7	2.3	3.9
-	Exclusion	696						

3.4 The content of GitHub README files

Table 3.3 demonstrates the distribution of categories based on the human annotation (column 3 on 'sections') and the README files in our sample (column 4 on 'files'). Based on manually annotated sections, the most frequent category is 'How' (58.4%), while the least frequent was 'Other' (1.4%). As mentioned previously, as part of the coding, our annotators also excluded non-English content that had not been detected by our automated filters (code '-'). The same applies to parts of README files that had been incorrectly detected as sections by our automated tooling.

Based on the consideration of files in our sample (fourth column of Table 3.3), 97% of the files contain at least one section describing the 'What' of the repository and 88.5% offer some 'How' content. Other categories, such as 'Contribution', 'Why', and 'Who', are much less common.

Table 3.4: Quantity of codes per section

# Codes	# Sections
5	2
4	6
3	40
2	498
1	3,680
Total	4,226

The last five columns of Table 3.3 demonstrate the distribution of codes across various file types (e.g., end-user applications, libraries). The most common code among all file types is 'How' while 'What' and 'References' are common in all file types except README files related to 'user interfaces'. Further, learning related resources such as assignments and tutorials rarely contain information related to 'When' and 'Contribution'.

Further, we report the distribution of number of codes across the sections of GitHub README files in our sample (Table 3.4). The sections that are annotated using four or five codes mostly stem from README files that only contain a single section. Interestingly, the majority of these files include 'What', 'Who', and 'References'. Also, 92% of the sections which are annotated using three codes include 'What'. Unsurprisingly, the most popular combination of two codes was 'How' and 'References', enabling access to additional information when learning 'how to use the project'. These relationships are further explored in the following section.

3.4.1 Relations between codes

As with any qualitative coding schema, there may be some overlap between the different types of sections outlined in our coding reference (cf. Table 3.2). For example, API documentation, which the coding reference shows as an example for 'References' is often also related to 'How' or could be related to 'Contribution'. To systematically investigate the overlap between different

Table 3.5: Association rules at section level

Rule			Support	Confidence
{Why, How}	⇒	{What}	0.002	1.00
{Why, References}	⇒	{What}	0.003	0.93

Table 3.6: Association rules at file level

Rule			Support	Confidence
{Who}	⇒	{What}	0.52	0.98
{How, References}	⇒	{What}	0.54	0.98
{References}	⇒	{What}	0.59	0.97
{How}	⇒	{What}	0.86	0.97
{References}	⇒	{How}	0.55	0.91
{What, References}	⇒	{How}	0.54	0.91
{What}	⇒	{How}	0.86	0.89

section types based on the manually annotated data, we applied association rule learning [5] to our data using the `arules` package in R. To find interesting rules, we grouped the data both by sections (i.e., each section is a transaction) and by files (i.e., each file is a transaction).

Table 3.5 shows the extracted rules at section level. We only consider rules with a support of at least 0.0013 (i.e., the rule must apply to at least five sections) and a confidence of at least 0.8. Due to the small number of sections for which we assigned more than one code, only two rules were extracted: Sections that discuss the 'Why' and 'How' are likely to also contain information on the 'What'. Similarly, sections that discuss the 'Why' of a project and contain 'References' are also likely to contain information on the 'What'.

At file level, we were able to find more rules, see Table 3.6. For these rules, we used a minimum support of 0.5 and a minimum confidence of 0.8. We chose a minimum support of 0.5 to limit the number of rules to the most prevalent ones which are supported at least by half of the README files in our dataset. The rules extracted with these parameters all imply 'What' or 'How' content to be present in a README file. For example, we have a 98% confidence that a file that contains information about 'Who' also contains information about

the ‘What’ of a project. This rule is supported by 52% of the README files in our dataset.

3.4.2 Examples

In this section, we present an example for each of the categories to illustrate the different codes.

What. The leading section of the GitHub README file of the `ParallelGit` repository[15] by GitHub user `jmilleralpine` is a simple example of a section that we would categorize into the ‘What’ category. The section header simply restates the project name (“ParallelGit”) and is followed by this brief description: “A high performance Java JDK 7 nio in-memory filesystem for Git.” Since this is an introduction to the project, we assign the code ‘What’.

Why. The README file of the same repository (`ParallelGit`) also contains a section with the heading “Project purpose explained” which we categorize into the ‘Why’ category. This section starts with a list of four bullet points outlining useful features of Git, followed by a brief discussion of the “lack of high level API to efficiently communicate with a Git repository”. The README file then goes on to explain that “ParallelGit is a layer between application logic and Git. It abstracts away Git’s low level object manipulation details and provides a friendly interface which extends the Java 7 NIO filesystem API.” Since this section describes the purpose of the project and motivates the need for it, we assign the code ‘Why’.

How. The same README file also contains a section with the heading “Basic usages”, which we classify into the ‘How’ category. It provides two short code snippets of seven and eight lines, respectively, which illustrate the use cases of “Copy a file from repository to hard drive” and “Copy a file to repos-

[15] `https://github.com/jmilleralpine/ParallelGit`

itory and commit". We assign the code 'How' because this section explains how to run the software.

When. An example of a section discussing the 'When' aspect of a project is given by the section with the heading "Caveats" of the `Sandstorm` repository[16] by GitHub user `solomance`. The project is a self-hostable web app platform. In its "Caveats" section, the README file states "Sandstorm is in early beta. Lots of features are not done yet, and more review needs to be done before relying on it for mission-critical tasks. That said, we use it ourselves to get work done every day, and we hope you'll find it useful as well." Since this section describes the project status, we assign the code 'When'.

Who. Going back to the README file of the `ParallelGit` repository, it concludes with a section with the heading "License" and the following text: "This project is licensed under Apache License, Version 2.0." A link to the license text is also included. We categorized this section under 'Who' since it contains licence information (see Table 3.2).

References. The previously mentioned README file of the `Sandstorm` repository also contains sections that we categorized as 'References', e.g., the section with the heading "Using Sandstorm". This section only contains the statement "See the overview in the Sandstorm documentation" which links to more comprehensive documentation hosted on `https://docs.sandstorm.io/`. We assign the code 'References' since the section does not contain any useful content apart from the link to more information. This section showcases one of the challenges of classifying the content of sections contained in GitHub README files: While the section header suggests that the section contains 'How' information, the body of the section reveals that it simply contains a link.

[16]`https://github.com/solomance/sandstorm`

Contribution. The README file of `Sandstorm` also contains a section with the heading "Contribute" which we categorized under 'Contribution'. The section states "Want to help? See our community page or get on our discussion group and let us know!" and contains links to a community page hosted on `https://sandstorm.io/` as well as a discussion group hosted on Google Groups.[17] We assign the code 'Contribution' rather than 'References' since this section contains information other than links, i.e., the different ways in which contributions can be made. Arguably, this is a corner case in which the code 'References' would also be justifiable.

Other. An example of a section that we were not able to categorize using any of the previous seven categories is the last section in the README file of the `Blackjack` repository[18] by GitHub user `ChadLactaoen`. The section does not contain any content and simply consists of the section heading "Have fun!" In this case, the section feature of GitHub markdown was used for highlighting rather than for structuring the content of the README file. We therefore categorized the section as 'Other'.

> **Finding 1:** Section content of GitHub README files can be categorized into eight types, with the 'What' and 'How' content types being very common and information on project status being rare.

3.5 A GitHub README Content Classifier

In this section, we describe our automated classification approach for classifying GitHub README content. We first describe the overall framework of our approach and then explain each of its steps. For the development of this classifier, we use the set of sections associated with one of 8 classes along with sections labeled 'Exclusion', and split the dataset into two, a *development* set

[17] `https://groups.google.com/`
[18] `https://github.com/ChadLactaoen/Blackjack`

comprising 25% of the data, and an *evaluation* set comprising 75% of the data. We analyze and use the *development* set to design features for the classifier, such as heuristics based on language patterns (see Section 3.5.2.2). The *evaluation* set is the hold out set that is used for evaluation of the classifier through ten-fold cross-validation. A similar process of dividing a dataset into two – one for manual analysis for feature identification, and another for evaluation – has been done in prior studies (e.g., [152]) to improve reliability of reported results. Our code, dataset, along with scripts for the experiments as well as a README file containing information on how to use them are available at `https://github.com/gprana/READMEClassifier`

3.5.1 Overall Framework

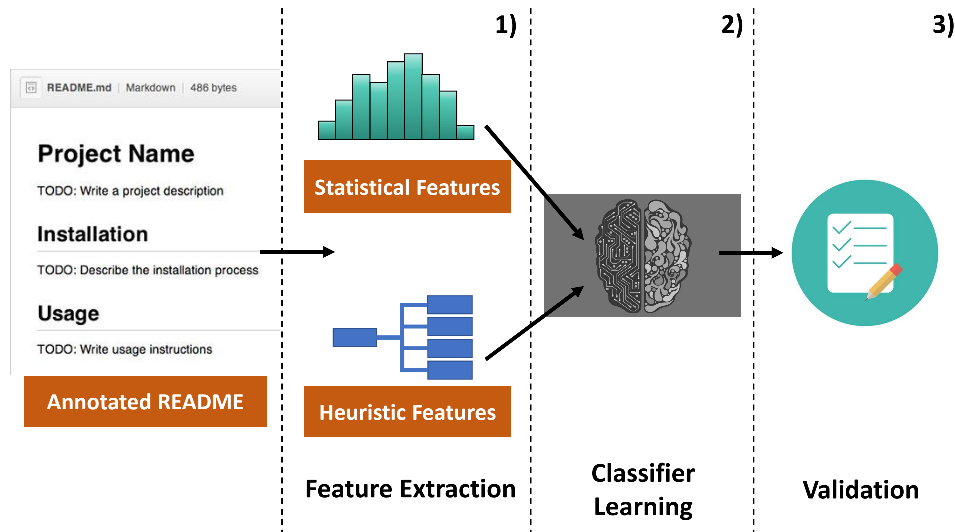

Figure 3.3: The overall framework of our automated GitHub README content classifier.

We present the overall framework of our automated classification approach in Figure 3.3. The framework consists of the following steps:

1. **Feature Extraction:** From each section of the annotated GitHub README files, we extract meaningful features that can identify categories of a section's content. We extract statistical and heuristic features. These features are output to the next step for learning.

2. **Classifier Learning:** Using features from the previous step, we learn a classifier that can identify the categories that the content of each section belongs to. Since each section can belong to many categories, we use a multi-label classifier, which can output several categories for each section.

3. **Validation:** To choose our classifier setting, we need to validate our classifier performance on a hold out set. We experiment with different settings and pick the classifier that performs the best on the hold out set.

We explain details of the above steps in the next subsections.

3.5.2 Feature Extraction

From the content of each section, we extract two sets of features: statistical features and heuristic features.

3.5.2.1 Statistical Features

These features compute word statistics of a README section. These features are constructed from combination of both heading and content of the section. To construct these features, the section's content and heading are first preprocessed. We perform two preprocessings: *content abstraction* and *tokenization*. Content abstraction abstracts contents to their types. We abstract the following types of section content: *mailto* link, hyperlink, code block, image, and numbers. Each type is abstracted into a different string (*@abstr_mailto*, *@abstr_hyperlink*, *@abstr_code_section*, *@abstr_image* and *@abstr_number*, respectively). Such abstraction is performed since for classification, we are more interested in existence of those types in a section than its actual content. For example, existence of a source code block in a section may indicate that the section demonstrates usage of the project, regardless of the source code. With abstraction, all source code blocks are converted to the same string, and sub-

sequently, into the same statistical feature. This abstraction is followed by tokenization, which converts a section into its constituent words, and English stop word removal. For the stop word removal, we use the stop words provided by *scikit-learn* [157].

After preprocessing, we count the number of times a word appears in each section. This is called the Term Frequency (TF) of a word in a section. If there are n words that appear in the set of sections used for training the classifier (after preprocessing), we would have n statistical features for each section. If a word does not appear in a section, then its TF is zero. We also compute the Inverse Document Frequency (IDF) of a word. IDF of a word is defined as the reciprocal of the number of sections in which the word appears. We use a multiplication of TF and IDF as an information retrieval feature for a particular word.

3.5.2.2 Heuristic Features

There has been work such as Panichella et al. [152] which exploits recurrent linguistic patterns within a category of sentences to derive heuristics that can aid classification. Given this, we manually inspected the content of various sections in the *development* set to try to identify patterns that may be useful to distinguish each category. The following are the resulting heuristic features that we use for the classifier.

1. **Linguistic Patterns:** This is a binary feature that indicates whether a particular linguistic pattern exists in a section. We discover linguistic patterns by looking at words/phrases that either appear significantly more in one particular category or are relatively unique to a particular category. A linguistic pattern is tied to either a section's heading or content. A pattern for heading is matched only to the section's heading. Similarly, a pattern for content is matched only to the section's content.

There are 55 linguistic patterns that we identified.[19]

2. **Single-Word Non-English Heading:** This is a binary feature that indicates whether a section's heading is a single word non-English heading. An example is a method name, which may be used as heading in a section describing the method and usually belongs to the 'How' category. This check is performed by checking the word against the wordlist corpora from NLTK [14].

3. **Repository Name:** This is a binary feature that indicates whether any word in the repository name is used in a section's heading. This is based on the observation that the README section that provides an overview of the project likely contains common words from the project name. For example, a repository of a project called 'X' will contain 'X' in its name, and the README section providing an overview of the project may be given a heading along the lines of 'About X', 'Overview of X', or 'Why X'. This is different, for example, from README sections containing licence information or additional resources.

4. **Non-ASCII Content Text:** This is a binary feature that indicates whether a section contains any non-ASCII character. It is based on the observation that README sections containing text written in non-ASCII characters tend to be categorized as 'Exclusion', although they often also contain parts (e.g., technical terms or numbers) written in ASCII characters.

3.5.3 Classifier Learning

Given the set of features from the previous step, we construct a multi-label classifier that can automatically categorize new README sections. We use

[19] The linguistic patterns are available in `https://github.com/gprana/READMEClassifier/blob/master/doc/Patterns.ods`.

a binary relevance method for multi-label classification [119]. This method transforms the problem of multi-label classification into a set of binary classifications, with each binary classification performed for one label independently from the other labels. Due to the small number of entries in the 'Why' category, combined with the fact that a large proportion of content in this category is also assigned to the 'What' category, we combined the two categories. We therefore ended up with eight categories including 'Exclusion', and subsequently created eight binary classifiers, each for a particular category.

A binary classifier for a particular label considers an instance that contains the label as a positive instance, otherwise it is a negative instance. As such, the training set for the binary classifier is often imbalanced. Thus, we balance the training set by performing oversampling. In this oversampling, we duplicate instances of minority classes and make sure that each instance is duplicated roughly until we have the same number of positive and negative instances in the set.

3.5.4 Validation

In this step, we determine the classifier setting by performing ten-fold cross validation. The setting that leads to the highest classifier performance is selected as final setting.

3.6 Evaluation of the Classifier

We conduct experiments with our SVM-based classifier on the dataset annotated in Section 3.4. We evaluate the classifier on the *evaluation* set using ten-fold cross validation. We follow our framework in Section 3.5 to construct our classifier. For evaluation, the TF-IDF vocabulary is constructed from the *evaluation* set, and is not shared with the *development* set. The size of this vocabulary created from the *evaluation* set is 14,248. We experiment with

the following classification algorithms: Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR), Naive Bayes (NB), and k-Nearest Neighbors (kNN). We use implementations of the classification algorithms from *scikit-learn* [157]. To evaluate the usefulness of the classification, we used the automatically determined classes to label sections in GitHub README files using badges and showed files with and without these badges to twenty software professionals.

3.6.1 Evaluation metric

We measure the classification performance in terms of F1 score. F1 score for multi-label classification is defined below.

$$F1 = \frac{\sum_{l \in L} w_l \times F1_l}{|L|}$$

$$F1_l = \frac{2 \times Precision_l \times Recall_l}{Precision_l + Recall_l}$$

where w_l is the proportion of the actual label l in all predicted data. $F1_l$ is the F1 score for label l, L is the set of labels, $Precision_l$ is precision for label l, and $Recall_l$ is the recall for label l. When computing precision/recall for label l, an instance having label l is considered as a positive instance, otherwise it is a negative instance. Precision is the proportion of predicted positive instances that are actually positive while recall is the proportion of actual positive instances that are predicted as positive.

For this work we consider both precision and recall as equally important. Taking into account that each section can have a different mix of content, our goal is to maximize completeness of the label set assigned to a section while avoiding clutter that can result from assigning less relevant labels.

Table 3.7: Results for Different Classifiers

Classifier	$F1$
SVM	0.746
RF	0.696
NB	0.518
LR	0.739
kNN	0.588

Table 3.8: Effectiveness of Our SVM-based Classifier

Category	F1	Precision	Recall
What and Why	0.615	0.627	0.604
How	0.861	0.849	0.874
When	0.676	0.669	0.683
Who	0.758	0.810	0.711
References	0.605	0.606	0.603
Contribution	0.814	0.857	0.774
Other	0.303	0.212	0.537
Exclusion	0.674	0.596	0.775
Overall	0.746	0.742	0.759

3.6.2 Evaluation results

The results of our evaluation are shown in Table 3.7. Our experimental results show that our SVM-based classifier can achieve an F1 score of 0.746 on the *evaluation* set using ten-fold cross validation. We also experiment with using SMOTE [30] on the best performing (SVM-based) classifier to compare its effectiveness with the oversampling approach, and found that it resulted in a lower F1 of 0.738.

The per category F1 obtained from the SVM-based classifier is shown in Table 3.8.

In addition to F1, we measured the performance of our classification using Kappa [108], ROC AUC [55], and MCC [19]. Our classifier can achieve a weighted average Kappa of 0.831, a weighted average ROC AUC of 0.957, and a weighted average MCC of 0.844. As prior work (e.g., [115, 161, 174, 220]) consider F-measure and/or AUC of 0.7 or higher to be reasonable, we believe

the evaluation result demonstrates that the SVM-based classifier design has sufficiently good performance.

Finding 2: We can automatically classify content of sections in GitHub README files with F1 of 0.746

3.6.3 Speed

We evaluate the speed of the best performing SVM-based classifier using a test machine with the following specifications: Intel Core i7-4710HQ 2.50 GHz CPU, 16 GB RAM laptop with SSD storage and Windows 10 64-bit. For this part of the evaluation, input data comprise the combined set README files from development and evaluation sets. We find that training of the classifier on this combined set takes 181 seconds. Afterwards, the classifier is able to label sections in a given input README file in less than a second. This indicates that the classifier is fast enough for practical use.

3.6.4 Multi-category sections vs. single-category sections

We expect that classifying multi-category sections is harder than classifying single-category sections. To confirm this, we exclude sections that belong to more than one category. We perform a similar experiment using ten-fold cross validation. Our experimental results show that our SVM-based classifier achieves an F1 score of 0.773, which confirms that classifying single-category sections is indeed easier, although not by a significant margin.

3.6.5 Usefulness of statistical vs. heuristic features

To investigate the value of a set of features, we remove the set and observe the classifier performance after such removal. Table 3.9 shows the classifier performance when we remove different sets of features. We observe performance reduction when removing any set of features. Thus, all sets of features are

Table 3.9: Contribution of Different Sets of Features

Set of Features Used	F1
Only Heuristic	0.584
Only Statistical	0.706

valuable for classifying README sections. Among the sets of features, the statistical features are more important since their removal reduces F1 far more as compared to removing heuristic features.

3.6.6 Usefulness of particular features

We are also interested in identifying which particular feature is more useful when predicting different categories. Using an SVM classifier, usefulness of a feature can be estimated based on the weight that the classifier assigns to the feature. For each category in the testing data, we consider an instance belonging to the category as a positive instance, otherwise it is a negative instance. We learn an SVM classifier to get the weight of each feature. To capture significantly important features, we perform the Scott-Knott ESD (Effect Size Difference) test [196]. For the purpose of this test, we perform ten times ten-fold cross validation where each cross validation generates different sets. Thus, for each category and feature pair, we have 100 weight samples. We average the weights and run Scott-Knott ESD test on the top-5 features' weights. We present the result for each category in Figure 3.4. Features grouped by the same color are considered to have a negligible difference and thus have the same importance.

Based on the observation, heuristics based on sections' headings appear to be useful in predicting categories. For example, *heur_h_k_012* (check whether a lower cased heading contains the string 'objective') is the second most useful features for predicting the 'What and Why' category, while *heur_h_k_006* (check whether a lower cased heading contains the string 'contrib') is the third most useful feature for predicting the 'Contribution' category. For the 'Who'

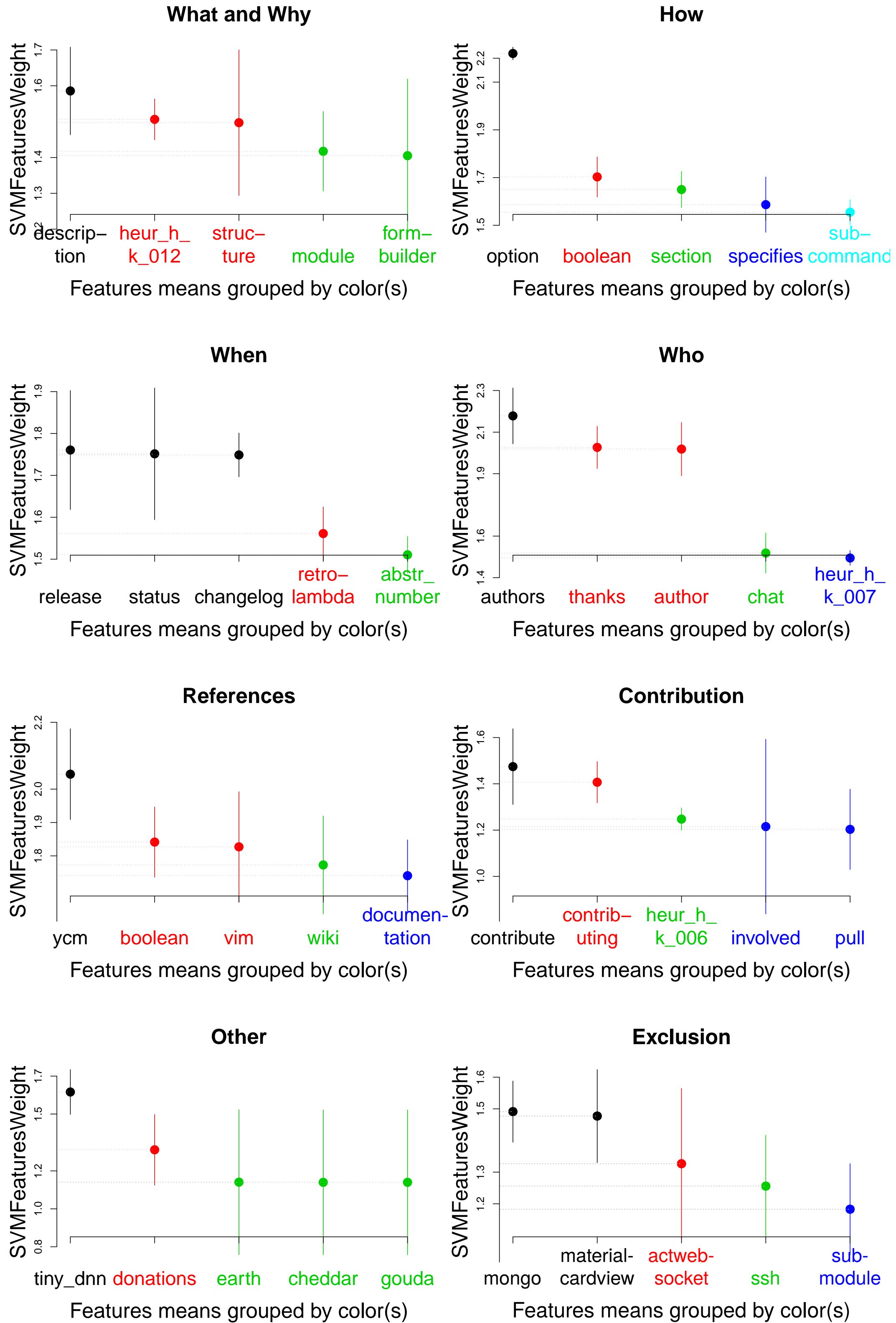

Figure 3.4: Top Features for Each Category. Features starting with *heur_* refer to heuristic features while the remaining features refer to statistical features (see Section 4.2).

category, *heur_h_k_007* (check whether a lower cased heading contains 'credit') is the fifth most useful feature for prediction. Abstraction also appears to be useful, with *@abstr_number* being the fifth ranking feature for predicting the 'When' category. A possible reason is that the 'When' category covers version history, project plans, and project roadmap, which often contain version number, year, or other numbers.

Finding 3: Overall, statistical features are more useful than heuristics, but heuristics based on section headings are useful to predict certain categories

3.6.7 Perceived usefulness of automatically labeling sections in GitHub README files

A potential use case for our work is to automatically label sections in GitHub README files. To evaluate the perceived usefulness of such an effort, we conducted a survey with 20 professional software developers (19 indicated to develop software as part of their job, 1 indicated to be an IT support specialist). We recruited participants using Amazon Mechanical Turk, specifying "Employment Industry - Software & IT Services" as required qualification.

As part of the survey, we showed each participant two versions of a randomly selected GitHub README file which we sampled using the criteria listed in Table 3.1. Note that the README files used for the survey were 'unseen' files, i.e., files that had not been used as part of the previously introduced *development* or *evaluation* sets. We prepared two README files that we selected using this sampling strategy by producing two versions of each file: one version was the original README file, the other version used badges [207] next to each section header to indicate the labels that our classifier had automatically assigned to the section. Table 3.10 shows the questions asked in the survey and examples of our prepared README files are available in Figure 3.5

Questions? content who content references

Open an Issue and let's chat!

Other forkable themes content references

You can use the Quick Start workflow with other themes that are set up to be forked too! Here are some of my favorites:

- Hyde by MDO
- Lanyon by MDO
- mojombo.github.io by Tom Preston-Werner
- Left by Zach Holman
- Minimal Mistakes by Michael Rose
- Skinny Bones by Michael Rose

Credits content who

- Jekyll - Thanks to its creators, contributors and maintainers.
- SVG icons - Thanks, Neil Orange Peel. They're beautiful.
- Solarized Light Pygments - Thanks, Edward.
- Joel Glovier - Great Jekyll articles. I used Joel's feed.xml in this repository.
- David Furnes, Jon Uy, Luke Patton - Thanks for the design/code reviews.
- Bart Kiers, Florian Simon, Henry Stanley, Hun Jae Lee, Javier Cejudo, Peter Etelej, Ben Abbott, Ray Nicholus, Erin Grand, Léo Colombaro, Dean Attali, Clayton Errington, Colton Fitzgerald, Trace Mayer - Thanks for your fantastic contributions to the project!

Contributing content contribution

Issues and Pull Requests are greatly appreciated. If you've never contributed to an open source project before I'm more than happy to walk you through how to create a pull request.

You can start by opening an issue describing the problem that you're looking to resolve and we'll go from there.

I want to keep Jekyll Now as minimal as possible. Every line of code should be one that's useful to 90% of the people using it. Please bear that in mind when submitting feature requests. If it's not something that most people will use, it probably won't get merged. 💂

Figure 3.5: An excerpt from a GitHub README file with visual labels, original version at `https://github.com/alt-blog/alt-blog.github.io`

and online.[20]

All participants indicated to have been developing software for several years, with a median of five years development experience (minimum: 2 years). All but two participants indicated having a GitHub account and having contributed to more than 20 repositories on average. Only 4 of the 20 participants indicated to have never contributed to a GitHub README file.

Table 3.11 shows the results we obtained about the perceived usefulness of the automated labeling of sections. The majority of participants (60%)

[20] Original: `https://github.com/readmes/alt-blog.github.io/blob/master/README1.md`, Modified: `https://github.com/readmes/alt-blog.github.io/blob/master/README2.md`

Table 3.10: Questions asked in the survey to determine perceived usefulness of automatically generated section labels

1	Is developing software part of your job?
2	What is your job title?
3	For how many years have you been developing software?
4	What is your area of software development?
5	Do you have a GitHub account?
6	Approximately how many repositories have you contributed to on GitHub?
7	Have you ever contributed to the GitHub README file for a repository?
8	What content do you expect to find in the README file of a GitHub repository?
9	What single piece of information would you consider most important to be included in a GitHub README file?
10	Is your decision to use or contribute to a GitHub project influenced by the availability of README files?
11	Please take a look at the following two README files. Which one makes it easier to discover relevant information, in your opinion? Note that only the badges next to sections titles are different.
12	Please justify your answer
13	Do you have any further comments about GitHub README files or this survey?

indicated that the files with our labels made it easier to discover relevant information, some participants did not have a preference, and only 2 participants preferred the unlabeled file. In general participants liked the labels, e.g., one participant wrote "I really like the Who, what, where, and why tags. It makes it easier to find relevant information when I only need to look for a certain section." Similarly another participant noted: "The what/when/how labels allow easier access to the information I am looking for." On the negative side, a minority of participants thought that the labels were not necessary: "the

Table 3.11: Survey results about the perceived usefulness of automatically labeled GitHub README sections

prefer labeled file	12
neutral	6
prefer unlabeled file	2
sum	20

extra buttons aren't really needed".

Finding 4: The majority of participants perceives the automated labeling of sections based on our classifier to ease information discovery in GitHub README files

3.7 Implications

The ultimate goal of our work is to enable the owners of software repositories on sites such as GitHub to improve the quality of their documentation, and to make it easier for the users of the software held in these repositories to find the information they need.

The eight categories of GitHub README file content that emerged from our qualitative analysis build a point of reference for the content of such README files. These categories can help repository owners understand what content is typically included in a README file, i.e., what readers of a README file will expect to find. In this way, the categories can serve as a guideline for a README file, both for developers who are starting a new project (or who are starting the documentation for an existing project) and developers who want to evaluate the quality of their README file. Even if all the content is in place, our coding reference provides a guide on how to organize a README file.

In addition, the categories along with their frequency information that we report in this chapter highlight opportunities for repository owners to stand out among a large crowd of similar repositories. For example, we found that only about a quarter of the README files in our sample contain information on the 'Why' of a repository. Thus, including information on the purpose of a project is a way for repository owners to differentiate their work from that of others. It is interesting to note that out of all the kinds of content that GitHub recommends to include in a README file (cf. Introduction), 'Why' is

the one that is the least represented in the README files of the repositories in our sample.

In a similar way, README content that refers to the 'When' of a project, i.e., the project's current status, is rare in our sample. In order to instill confidence in its users that they are dealing with a mature software project and to possibly attract users to contribute to a project, this information is important. However, our qualitative analysis found that less than a quarter of the repositories in our random sample included 'When' information.

The ratio of repositories containing information about how to contribute was slightly higher (109/393), yet surprisingly low given that all of the repositories in our sample make their source code available to the public. Given recent research on the barriers experienced by developers interested in joining open source projects [195], our findings provide another piece of evidence that software projects have room for improvement when it comes to making a good first impression [57] and explaining how developers can contribute.

The classifier we have developed can automate the task of analyzing the content of a README file according to our coding reference, a task that would otherwise be tedious and time-consuming. Our classifier can take any GitHub README file as input and classify its content according to our codes with reasonable precision and recall.

In addition to automatically classifying the content, our classifier could enable semi-structured access to the often unstructured information contained in a GitHub README file. For example, users particularly interested in finding mature projects could automatically be brought to the 'When' sections of a README file, and developers looking to contribute to open source could be shown the 'Contribution' guidelines of a repository.

The results from our survey show evidence which indicates that visually labeling sections using the labels predicted by our classifier can make it easier to find information in GitHub README files: The majority of participants

perceived the automated labeling of sections based on our classifier to ease information discovery. Visually labeling sections is only one use case of the classifier: Our classifier could also easily be used to help organize README files, e.g., by imposing a certain order in which sections should appear in a README file. README sections that have been detected as discussing the 'What' and 'Why' of a project could automatically be moved to the beginning of a README file, followed by sections discussing the 'How'.

Our analysis of the usefulness of features for predicting the categories of a section implies that heuristic features on the sections' headings are useful, and are better suited than heuristic features on the sections' contents. This is apparent from the fact that none of the heuristic features for sections' contents are ranked among the top-5 most useful features for any of the categories. This suggests that the vocabulary commonly used in section headings is more uniform than that used in section content. However, we note that the 4,226 sections in our dataset use 3,080 distinct headings, i.e., only few of the sections share the same heading.

3.8 Threats to Validity

Similar to other empirical studies, there are several threats to the validity of our results.

Threats to the construct validity correspond to the appropriateness of the evaluation metrics. We use F1 as our evaluation metric. F1 has been used in many software engineering tasks that require classification [96, 165, 141, 26, 164]. Thus, we believe threats to construct validity are minimal. In our survey, we measured perceived usefulness of the visual labels added to GitHub README files, which may not correspond to actual usefulness in a software development task. Future work will have to investigate this in more detail.

Threats to the internal validity compromise our confidence in establishing

a relationship between the independent and dependent variables. It is possible that we introduced bias during the manual annotation of sections from GitHub README files. We tried to mitigate this threat by using two annotators, and by manually resolving all cases in which the two annotators disagreed. We did however notice a small number of cases where annotators mistakenly treated non-sections (e.g., content that had been commented out) as sections.

Threats to external validity correspond to the ability to generalize our results. While our sample of 393 GitHub README files is statistically representative, it is plausible that a different sample of files would have generated different results. We can also not claim generalizability to any other format of software documentation. We excluded README files that were small (less than 2 KB in size), README files that belonged to repositories not used for software development, and README files not in English. Different filtering criteria might have led to different results. Our findings may also have been impacted by our decision to divide README files into sections. A different way of dividing README files (e.g., by paragraphs or sentences) might also have produced different results. Our survey was answered by twenty software professionals. We cannot claim that we have captured all possible opinions regarding the usefulness of the visual labels. All survey participants were ultimately self-selected individuals within our target populations, and individuals who did not respond to our invitations may have different views on some of the questions that we asked. Also, creating visual labels is only one use case of our classifier, and we cannot make claims of the usefulness of other applications based on our survey results.

3.9 Conclusions and Future Work

A README file is often the first document that a user sees when they encounter a new software repository. README files are essential in shaping the

first impression of a repository and in documenting a software project. Despite their important role, we lack a systematic understanding of the content of README files as well as tools that can automate the discovery of relevant information contained in them.

In this chapter, we have reported on a qualitative study which involved the manual annotation of 4,226 sections from 393 README files for repositories hosted on GitHub. We identified eight different kinds of content, and found that information regarding the 'What' and 'How' of a repository is common while information on the status of a project is rare. We then designed a classifier and a set of features to automatically predict the categories of sections in README files. Our classifier achieved an F1 score of 0.746 and we found that the most useful features for classifying the content of README files were often related to particular keywords. To evaluate the usefulness of the classification, we used the automatically determined classes to label sections in GitHub README files using badges and showed files with and without these badges to twenty software professionals. The majority of participants perceived the automated labeling of sections based on our classifier to ease information discovery.

Our findings provide a point of reference for repository owners against which they can model and evaluate their README files, ultimately leading to an improvement in the quality of software documentation. Our classifier will help automate these tasks and make it easier for users and owners of repositories to discover relevant information.

In addition to improving the precision and recall of our classifier, our future work lies in exploring the potential of the classifier to enable a more structured approach to searching and navigating GitHub README files. In particular, we plan to employ the classifier in a search interface for GitHub repositories and we will explore the feasibility of automatically reorganizing the documentation contained in GitHub README files using the structure that emerged from

our qualitative analysis.

Chapter 4

Study on Vulnerabilities in Open Source Software Dependencies

4.1 Introduction

Modern software is typically built using a large amount of third-party code in the form of external libraries to save development time. Such third-party components are often used as is [116], and even for trivial functions, developers often choose to use an external library instead of writing their own code [2]. Centralized repositories (such as Maven Central and PyPI) and their associated dependency management tools make it easy for software developers to download and include open-source libraries in their projects, further improving the developers' productivity.

However, such third-party libraries may contain varying amount of security vulnerabilities. While developers may get their own code reviewed by peers or checked for bugs or security issues by using static analysis tools [99], Kula et al. found that developers often do not review the security of third-party libraries, citing it as extra effort [102]. Since a software project may depend on a number

of open-source libraries, which may in turn depend on many other libraries in a complex package dependency network, analysis on a software project's entire dependency tree can become very complex. Unchecked project dependencies may introduce security vulnerabilities into the resulting software, which may be hard to detect. An example of this is the buffer overread in OpenSSL library that resulted in Heartbleed vulnerability [50], which was introduced in 2012 but remained undetected until 2014. Another high-profile example is the unpatched CVE-2017-5638 vulnerability in Apache Struts that resulted in the 2017 Equifax data breach. More recently, CVE-2018-1000006 vulnerability that was discovered in the popular Electron framework in January 2018 affected a number of Windows applications built using the framework, such as Skype and Slack.

As the usage of open-source libraries grows, it becomes increasingly important to understand the risks associated with vulnerabilities in the libraries. This motivates us to investigate the prevalence of vulnerabilities in open-source libraries, the types and persistence of the vulnerabilities, along with relationships between their prevalence and project as well as commit attributes. Such an investigation can answer several open questions, for example: *What are common types of dependency vulnerabilities library users should be aware of? Is it sufficient for a library developer to fix vulnerabilities in their library and release a new version as soon as possible? How often are vulnerable dependencies left unchanged due to actual lack of newer versions? How effectively can we reduce risk due to vulnerable dependencies by adding more personnel to the project?* We believe that the result of such an investigation would be helpful to library users, library developers, and security researchers in a number of ways. Library developers can benefit from understanding the prevalence of persistent vulnerabilities as well as the prevalence of outdated dependencies, as they can signify the need to encourage updates and make updates easier. Library users can benefit from understanding common types of vulnerabili-

ties since such knowledge can help them to anticipate and guard against these common vulnerabilities. This is important as vulnerabilities in libraries may not become publicly known immediately and there may also be latency before library developers provide a fix. They can also benefit from understanding whether factors such as number and experience of contributors translate into better handling of vulnerable dependencies, since this can affect personnel allocation decisions, among others. In addition, the result of such investigation can also help researchers to identify directions of research that are more likely to benefit the widest range of software projects.

There are several open-source tools such as *OWASP Dependency Check*[1], *Bundler-audit*[2], and *RetireJS*[3] that can assist development teams to check for publicly-known security vulnerabilities in their open-source dependencies. Since November 2017, GitHub has also provided a service[4] that scans dependencies of a given project in several supported languages for publicly known vulnerabilities. Beyond this, several vendors, such as *Sonatype*, *Synopsys*, *Veracode*, and *WhiteSource*, also offer software composition analysis (SCA) tools that can identify open-source libraries used in a given software project, vulnerabilities associated with those libraries (including those not yet in public vulnerability databases), associated licenses, and other metrics. Such SCA tools enable development teams to identify vulnerable dependencies and other potential issues such as outdated dependencies and license issues.

In this work, we use Veracode Software Composition Analysis (SCA) tool to perform an empirical study on a sample of projects and their associated commits on GitHub. We use Veracode SCA as it is available to us, includes a database of open-source libraries maintained by Veracode security researchers along with categorized list of associated vulnerabilities (as well as their severity scores), and supports the three languages investigated in this study (Java,

[1]https://www.owasp.org/index.php/OWASP_Dependency_Check
[2]https://github.com/rubysec/bundler-audit
[3]http://retirejs.github.io/retire.js/
[4]https://help.github.com/en/articles/about-security-alerts-for-vulnerable-dependencies

Python, and Ruby). This enables systematic investigation and comparison of detected vulnerabilities in sampled projects' open-source dependencies. For our dataset, we sampled 450 software projects on GitHub that are written in Java, Python, and Ruby and have at least 5 commits during the 1-year period between November 1, 2017 to October 31, 2018. Being larger and more diverse than datasets of earlier works on vulnerability dependency usage [21, 110, 156]), our dataset enables better generalizability of analysis results. We subsequently checked out all commits made to the sampled projects during the 1-year period, and used Veracode SCA to scan the complete project version after each commit. Afterwards, we analyzed the scan results, which include vulnerability details such as CVE identifier, type, and severity. We examined a variety of aspects related to characteristics of the discovered vulnerabilities in the sampled projects' open-source dependencies, including common types, frequency, persistence, as well as the relationship between the vulnerabilities with project as well as commit attributes. In summary, we intend to answer the following research questions:

- ***RQ1****: What are the common types and prevalence of dependency vulnerabilities in open-source software, and how persistent are they?*

  Understanding common types, prevalence, and persistence of dependency vulnerabilities would help us assess the severity of the problem as well as shed light on ways to resolve or mitigate the vulnerabilities. This serves as our motivation to answer this research question. Among others, we found that such vulnerabilities are persistent and take months to fix. We also found common vulnerability types across languages.

- ***RQ2****: What are the relationships between vulnerabilities in a project's open source dependencies with the attributes of the project and its commits?*

  Many open-source developers and users hold the view that more reviewers result in improved software quality. This view is phrased as "many

eyes make all bugs shallow" by Eric Raymond and is known as Linus' Law [170] and has been investigated in several studies (e.g. [130, 131]). The argument is that larger size of community working on and using a particular software will make it more likely for any quality issues to be discovered and fixed. We are interested in examining whether this view holds true for dependency vulnerability count. Further, there has also been various works in vulnerability prediction (e.g. [140, 143, 228, 181, 224, 87]) that utilizes different types of metrics such as complexity, churn, and developer activity to predict vulnerability in the project's own code. Given this, we believe that it is worthwhile to examine possible correlations between some of the metrics with vulnerabilities resulting from the projects' open-source dependencies, in addition to comparing the correlation between vulnerabilities and the counts of project's direct and transitive dependencies. Direct dependencies refer to dependencies that are referenced by the project's code directly, while transitive dependencies refer to libraries that are referred to by other dependencies. We found that the vulnerability counts correlate more strongly with the project's total dependency counts compared to the project activity level, popularity, scale of commit, and experience level of the developer making a commit. This suggests, for example, that reducing the total number of dependencies (which may lack tests and have many dependencies on their own [2]) will be more effective in mitigating such vulnerabilities than recruiting additional developers into the project.

There have been several works focusing on the usage of vulnerable dependencies [21, 110, 156] as well as works that include discussion of vulnerable dependencies in context of library migration [46, 44, 102]. We expand on the earlier set of works by analyzing larger and more diverse set of software projects. In addition, the vulnerability details in the database of the Veracode SCA tool that we use enables investigation into some aspects not covered in

the above works, such as prevalence of different vulnerability types. Finally, we perform a scan on each commit made to the sampled projects within the 1-year observation period, enabling analyses related to persistence of the vulnerabilities and the correlation between vulnerabilities with commit attributes.

The chapter is structured as follows: Section 4.2 presents an overview of Veracode SCA tool used in our study. Section 4.3 discusses the dataset collection method, overview of the dataset, and our methodology. Section 4.4 presents the results of our empirical study. Section 4.5 discusses the implications of our findings to library users, developers, as well as researchers. Section 4.6 discusses threats to validity of our study. Section 4.7 concludes our work and presents future directions.

4.2 Overview of Veracode SCA

Veracode SCA[5] is a software composition analysis (SCA) tool from Veracode. SCA tools are typically used by developers and organizations to identify open-source components used by their software projects as well as various information associated with those components, including their respective licenses, known vulnerabilities, and latest available versions. Such tools help their users to prevent or mitigate security and legal issues, in addition to providing better visibility into their software projects.

Veracode SCA supports analysis in several languages (Java, Python, .NET, Ruby, JavaScript, PHP, Scala, Objective C, and Go), and works as follows: given a project code base, if necessary, it builds the project with the build system used by the project (e.g. Maven) and generates dependency graph from the result. It subsequently analyzes the graph to identify the open-source libraries used in the project. Afterwards, it matches the identified open-source libraries and their specific versions against a database containing information of open-source libraries obtained from variety of sources (e.g. Maven Central,

[5]https://www.veracode.com/products/software-composition-analysis

Ruby Gems, public sources of vulnerability information, as well as in-house research efforts). Based on this, Veracode SCA subsequently reports open-source libraries used by the project, the specific versions of the detected libraries, their licenses, associated vulnerabilities, as well as usage of outdated libraries, as shown in Figure 4.1. Veracode SCA includes static checking mechanism to aid library updates [59] as well as Security Graph Language [58], a domain-specific language that is designed to describe and represent vulnerabilities. The language supports efficient queries involving relations between open-source libraries, their file contents (such as methods and classes), and vulnerabilities.

Veracode SCA is also able to detect publicly-known vulnerabilities in Common Vulnerabilties and Exposures (CVE) list[6] in addition to a number of vulnerabilities that have not yet been assigned CVE identifiers. As of 27 June 2019, the Veracode SCA vulnerability database[7] contains 2,027,092 libraries from all supported languages (not counting different versions), and 11,364 distinct vulnerabilities. The library information are retrieved from various open source package repositories. For the languages used in this work (Java, Python, and Ruby), the Veracode SCA vulnerability database statistics are shown in Table 4.1.

Table 4.1: Veracode SCA vulnerability database information for languages used in this work. Note: Distinct vulnerability corresponds to a CVE for publicly-known vulnerabilities, or Veracode SCA artifact ID for non-publicly-known vulnerabilities.

Language	Libraries	Distinct Vulnerability	Source of library information
Java	240,015	1,484	*search.maven.org*, *repo1.maven.org* (for Maven)
Python	178,633	788	*pypi.python.org*
Ruby	138,082	648	*rubygems.org*

As source of vulnerability data, Veracode SCA makes use of both publicly-

[6]https://cve.mitre.org/

[7]https://sca.analysiscenter.veracode.com/vulnerability-database/search

known vulnerability information from National Vulnerability Database[8], as well as in-house research efforts to discover vulnerabilities that are not yet publicly known. Figure 4.2 provides high-level overview of the workflow. Identification of new vulnerabilities for inclusion into the database is achieved by Veracode security researchers through a variety of approaches, such as application of natural language processing and machine learning model to identify vulnerability-related commits and bug reports. The machine learning model [226] achieved precision of 0.83 and recall of 0.74 during validation at Veracode SCA production system in March 2017 - May 2017 and was able to detect more actual vulnerabilities than the number reported in CVE in the same period (349 vs 333). It outperformed the SVM-based classifier [158], one of the state-of-the-art approaches for vulnerability detection from commit messages as well as from bug reports. For instance, it achieved 54.55% higher precision with the same recall for commit messages. Beyond this, prior to publishing into the Veracode SCA database, each vulnerability is reviewed and researched by at least two Veracode security analysts. In addition to this, Veracode also keeps track of customer feedback regarding the vulnerability database. These factors support our confidence in the tool's detection capability. These factors support our confidence in the tool's detection capability.

By default, as part of its scan result, Veracode SCA reports Common Vulnerability Scoring System security score of vulnerabilities along with the following corresponding rating:

- 0.1 - 3.9 : Low
- 4.0 - 6.9 : Medium
- 7.0 - 8.9 : High
- 9.0 - 10.0 : Critical

[8]https://nvd.nist.gov/

PROJECT INVENTORY

Issues | Vulnerabilities | Libraries | Licenses

Matched Libraries

All languages | Direct only | Out-of-date | Has vulnerabilities

Search libraries | 157 libraries

Library	Version or (Commit) in use	Latest available	Vulnerabilities
excon	0.62.0	0.64.0	0 2 0
yard	0.9.16	0.9.20	0 0 1
puppet-syntax	2.4.1	2.5.0	0 0 0
beaker-vagrant	0.6.1	0.6.2	0 0 0

Figure 4.1: Part of result of a Veracode SCA scan

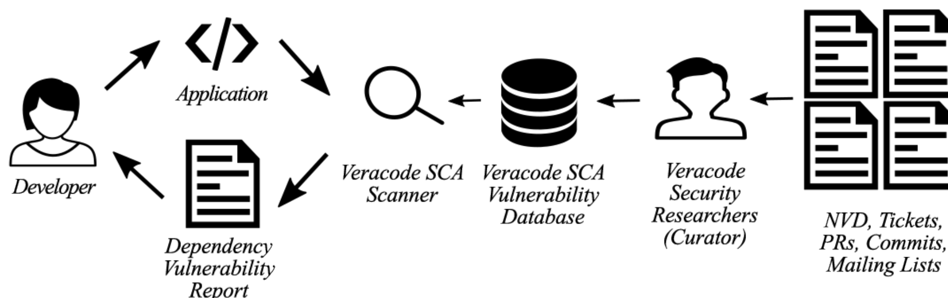

Figure 4.2: Overview of Veracode SCA workflow

Each detected vulnerability is also associated with at least one tag (such as "Authentication" or "Cross-site Scripting (XSS)"). The complete list of tags used in Veracode SCA database is shown in Table 4.2.

Overall, Veracode SCA's scan features and details in its scan results facilitate our analysis for characterizing the vulnerabilities in the open-source dependencies of the projects that we sampled. As our study involves multiple languages (Java, Python, and Ruby), Veracode SCA's language support also put it in an advantage compared to popular open-source alternatives such as *Bundler-audit* (which only supports Ruby Gems) or *OWASP Dependency Check* (which supports Java but has only experimental support for Ruby and

Table 4.2: List of vulnerability tags used by Veracode SCA

Authentication	Mass-assignment
Authorization	OS Command Injection
Buffer Overflows	Phishing attack
Business Logic Flaws	Race Conditions
Configuration	Remote DOS
Cross Site Scripting (XSS)	Remote Procedure Calls
Cryptography	Session Management
Data at Rest	Source code disclosure
Denial of Service	SQL Injection
EL execution	Transport Security
File I/O	Trojan Horse
Information Disclosure	XML Injection
Injection Vulnerabilities	XPath Injection
Man-in-the-middle	Other

Python).

4.3 Dataset & Methodology

4.3.1 Dataset Collection

We use GitHub as the source of software projects for this study. Since many GitHub repositories do not actually contain software projects [91], as starting point we used the *reaper* dataset from Munaiah et al. [138] which provides a list of repositories likely to contain software projects. We believe the benefit of performing sampling on this pre-filtered list of repositories outweighs the potential downside of missing newer repositories, and at the time the data collection began (December 2018) we were not aware of newer dataset of similar type. We set the following criteria for sampling the projects:

1. The project is written in Java, Python, or Ruby, based on information from the *reaper* dataset.

2. The project repository commit log lists at least five commits between November 1, 2017 and October 31, 2018.

3. The project satisfies the prerequisites to be scanned by the Veracode

SCA tool, i.e. its content indicates that it uses a build tool supported by Veracode SCA, and it is actually buildable. For Java projects, we focused on Maven projects to reduce the potential complexity of troubleshooting build issues. For Python projects, we look for the existence of one of the following files in the project's root directory: *setup.py*, *requirements.txt*, *requirements-dev.txt*, or *dev-requirements.txt*. For Ruby projects, we look for the existence of *Gemfile* in the project root directory.

The criteria are set to ensure that the resulting set of the sample projects comprises actively-maintained software projects written in popular languages, which should subsequently improve generalizability of our analysis results. In addition, the choice of selecting projects from multiple programming languages instead of collecting a larger set from a single language is meant to enable investigation into potential differences in characteristics of vulnerabilities in different languages.

After filtering for projects that match the criteria, we randomly sampled 450 software projects. This corresponds to 150 for each programming language (out of 462,182 Java projects, 331,883 Python projects, and 363,801 Ruby projects on *reporeaper*). Afterwards, we extracted the list of all commits made between November 1, 2017 and October 31, 2018. We subsequently scan the projects using Veracode SCA to identify its open-source dependencies as well as the type of each dependency (i.e. direct, transitive, or both). The statistics of the sampled projects are shown in Table 4.3. Table 3 shows that the number of transitive dependencies of the sampled projects are generally much higher than that of direct dependencies, consistent with observation of Decan et al. [45].

In addition to this, Figure 4.3 shows the relationship between commit author count and direct dependency count of the sample projects, while Figure 4.4 shows the relationship between commit author counts and transitive dependency counts. Table 4.4 shows the correlation (computed using Spearman's rank correlation test [192]) between commit author count and commit count,

as well as between commit author count and dependency counts. Following scale of interpretation of ρ used by Camilo et al. [22] ($\pm$ 0.00 - 0.30: Negligible, $\pm$ 0.30 - 0.50: Low, $\pm$ 0.50 - 0.70: Moderate, $\pm$ 0.70 - 0.90: High, and $\pm$ 0.90 - 1.00: Very high), we note that there is low to moderate correlation between commit author count and commit count, but no statistically significant correlation between commit author count and dependency count.

Table 4.3: Statistics of the sampled projects at latest commit in the observation period.

Metric		Java	Python	Ruby
Commits in target period	Min	5	5	5
	Max	4579	2471	1802
	Median	23.0	20.5	16.0
	Mean	104.6	76.4	56.9
	Std. dev.	371.4	196.7	165.9
Commit authors in target period	Min	1	1	1
	Max	22	51	43
	Median	2	3	2
	Mean	3.9	4.7	4.1
	Std. dev.	4.4	6.3	5.7
Direct open-source software (OSS) dependency	Min	0	0	0
	Max	81	29	99
	Mean	8.8	3.5	15.7
	Median	5	2	8
	Std. dev.	12.0	4.5	20.4
Transitive OSS dependency	Min	0	0	0
	Max	254	191	280
	Mean	29.6	7.0	53.2
	Median	9.5	1	43
	Std. dev.	44.1	18.2	49.0
Projects with no detected OSS dependency		16	35	2

Table 4.4: Correlation between commit author count and commit count, as well as between commit author count and dependency counts

Language	Number of Commits		Dependency Count			
			Direct		Transitive	
	ρ	p	ρ	p	ρ	p
Java	0.484	0.000	0.072	0.378	0.062	0.449
Python	0.587	0.000	0.080	0.333	-0.054	0.512
Ruby	0.367	0.000	-0.075	0.364	-0.047	0.565

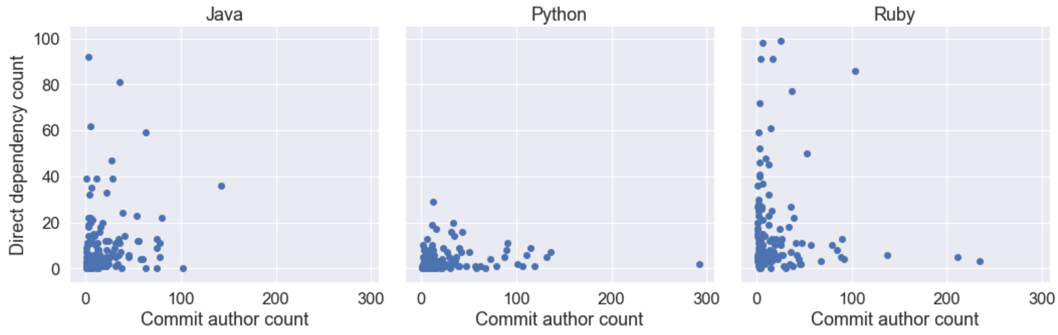

Figure 4.3: Relationship between sample projects' commit author count and direct dependency count

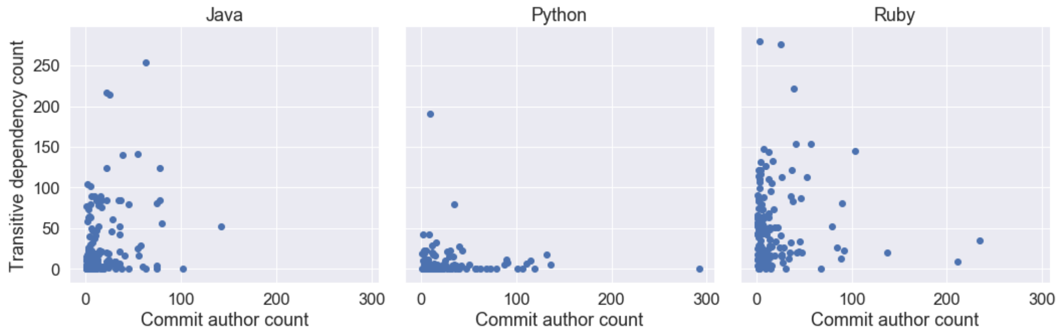

Figure 4.4: Relationship between sample projects' commit author count and transitive dependency count

4.3.2 Methodology

After selecting the GitHub projects and downloading their commit history, we checked out each commit and performed a scan on the full project versions after each commit using Veracode SCA agent. The tool reports total counts of direct and transitive open-source dependencies in the specific project version scanned, list of detected vulnerabilities (including description, severity score, and specific libraries that contain them), as well as other information such as license information of the libraries. We subsequently use the information on vulnerable open-source dependencies and their associated vulnerabilities for subsequent analyses.

For the purposes of counting distinct vulnerabilities, there are two cases to be considered: The first case is the vulnerabilities that have been assigned Common Vulnerabilities and Exposures (CVE) identifier. The CVE identifier points to a specific publicly-known vulnerability in the CVE list. The other

case relates to vulnerabilities that have not yet been assigned CVE identifier after their discovery by Veracode security researchers. Since the Veracode SCA vulnerability database assigns one artifact ID for each distinct vulnerability (with or without CVE), we use this artifact ID instead of CVE identifier. For subsequent analyses, we count a combination of software project, library version, and artifact ID as individual vulnerability instance.

In this work, we use "first commit" or "earliest commit" as a shorthand for first commit in the observation period (i.e. first commit in November 2017). Similarly, "last commit" or "latest commit" refers to latest commit in the observation period (i.e. latest commit in October 2018).

4.4 Empirical Study Results

In this section we discuss the results of our investigation into the characteristics of vulnerabilities in the sampled projects' open-source dependencies, as well as the vulnerabilities' relationship with project and commit attributes.

4.4.1 RQ1: What are the common types and prevalence of dependency vulnerabilities in open-source software, and how persistent are they?

4.4.1.1 Dependency vulnerability counts

Table 4.5 shows the distribution of the total counts of detected vulnerabilities in open-source dependencies of the sampled projects. The data shown is based on scan result at the time of the latest commit in the observed period, and is split into data on vulnerabilities with CVE (i.e. publicly-known vulnerabilities, including those for which CVE ID has been reserved at the time of scan) and vulnerabilities without CVE (i.e. non-publicly-known vulnerabilities). It shows that the Java sample set has the largest overall range and variation of

vulnerability counts, followed by the Ruby sample set.

Table 4.5: Overview of sample projects' vulnerability counts

Vulnerabilities with CVE					
Language	Min	Max	Mean	Median	Std.dev.
Java	0	98	9.0	1.0	15.6
Python	0	30	0.9	0.0	3.5
Ruby	0	42	4.4	1.0	7.2
Non-CVE Vulnerabilities					
Language	Min	Max	Mean	Median	Std.dev.
Java	0	19	1.9	0.0	3.3
Python	0	6	0.1	0.0	0.5
Ruby	0	31	3.0	1.0	5.2
Percentage of dependencies with vulnerability					
Language	Min	Max	Mean	Median	Std.dev.
Java	0.0	100.0	12.3	11.7	14.5
Python	0.0	100.0	7.8	0.0	18.4
Ruby	0.0	28.6	5.3	3.3	6.3

In addition to the total counts, we examine the breakdown of the vulnerabilities by dependency type. Specifically, we are interested in finding the average percentages of vulnerabilities that are associated with a project's direct dependencies, transitive dependencies, and dependencies that are used both directly and transitively. We perform this analysis at the latest commit of each project for which at least one dependency vulnerability is found. This gives us the most up-to-date information of the projects' vulnerability characteristics. The breakdown of vulnerability by dependency type is shown in Table 4.6. It shows that distribution of vulnerability counts by dependency type corresponds to relative proportion of dependency types. For Python projects, most of the dependency vulnerabilities are in the projects' direct dependencies, which are more visible to the project developers and more easily updated compared to transitive dependencies. On the other hand, given the higher percentage of vulnerabilities in Java and Ruby projects' transitive dependencies, developers using those languages will benefit more from careful scrutiny of their projects' transitive dependencies.

Table 4.6: Per-project vulnerability percentage distribution by dependency type. 'Both' denotes dependencies that are used both directly and transitively. Includes only projects with at least one vulnerability. Percentages of dependency by type ('Dep.') included for comparison.

		Java			Python			Ruby		
		CVE	Non-CVE	Dep.	CVE	Non-CVE	Dep.	CVE	Non-CVE	Dep.
Direct	Min	0.0	0.0	0.0	0.0	0.0	15.05	0.0	0.0	1.2
	Max	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	41.9
	Mean	30.3	36.7	29.2	85.0	50.0	51.4	10.0	16.3	18.2
	Median	4.0	19.5	15.7	100.0	50.0	44.4	0.0	0.0	19.6
	Std.dev	40.6	42.3	26.9	33.7	54.8	27.3	24.3	26.3	10.3
Transitive	Min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	54.6
	Max	100.0	100.0	100.0	100.0	100.0	85.0	100.0	100.0	98.8
	Mean	58.6	60.3	64.4	15.0	50.0	48.6	84.5	73.4	77.9
	Median	76.3	69.7	69.2	0.0	50.0	55.6	100.0	82.6	76.5
	Std.dev	42.8	41.6	23.4	33.7	54.8	27.3	27.1	31.8	11.7
Both	Min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Max	100.0	75.0	100.0	0.0	0.0	0.0	100.0	100.0	23.5
	Mean	11.1	3.1	6.5	0.0	0.0	0.0	5.5	10.3	3.9
	Median	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.7
	Std.dev	27.3	11.9	14.6	0.0	0.0	0.0	15.0	20.3	3.6

Finding 1: Proportion of vulnerability counts by dependency type varies by programming language, corresponding to relative proportion of dependency types

4.4.1.2 Most common dependency vulnerability types

To identify common types of dependency vulnerabilities in each language, we combine scan results of latest commits of each language's sample set and count all the detected vulnerabilities. For this analysis, we count each combination of library version, vulnerability, and project separately. That is, if a particular library version with two vulnerabilities of "Denial of Service" type is used by two projects, this is counted as four instances of "Denial of Service". If a project uses three libraries containing one "Denial of Service" type of vulnerability each, this is counted as three instances of "Denial of Service" vulnerability. We use tags associated with each vulnerability in the Veracode SCA database

for the categorization. The list of tags is shown in Table 4.2. Table 4.7 shows the total instances.

Table 4.7: Summarized count of dependency vulnerability instances at latest commit. Non-CVE vulnerabilities are identified by Veracode SCA artifact ID

		Java	Python	Ruby
Total vulnerability instances	CVE	1354	131	657
	Non-CVE	282	12	455
Distinct vulnerabilities (CVE / artifact ID)	CVE	212	51	80
	Non-CVE	56	9	74

Afterwards, we identify the tags associated with each vulnerability instance, count the total for each tag, and shortlist the ones with highest counts. Table 4.8 shows the top five results for each language. The result indicates some commonalities between the kinds of dependency vulnerabilities in each language, with “Denial of Service” and “Information Disclosure” being two common top issues across the three languages. This suggests that improvement of practices or tools to combat those vulnerability types (by both open-source library developers and security researchers) would bring significant benefits to a wide range of software projects.

Finding 2: “Denial of Service” and “Information Disclosure” are common across programming languages.

4.4.1.3 Distribution of severity scores

In addition to number of vulnerabilities, we are also interested in the severity of the vulnerabilities detected in the sampled projects’ open-source dependencies. Table 4.9 shows the distribution of severity according to the default rating scale, i.e. CVSS score of 0.1 - 3.9 : Low, 4.0 - 6.9 : Medium, 7.0 - 8.9 : High, 9.0 - 10.0 : Critical. Table 4.10 shows the distribution of the severity score for the top vulnerability types. The distribution of severity shows that most of the vulnerabilities in the sampled projects’ open-source dependencies are not critical. This is also the case for the two types of vulnerability that are

Table 4.8: Most common dependency vulnerability tags in each language. "CVE" and "non-CVE" indicate publicly-known and non-publicly-known vulnerabilities, respectively. Instance count and percentage denote count and percentage across the sample set of the programming language. Note that one vulnerability may have more than one tag.

Language	Tag	Total instances	% of all instances	CVE vuln. instances	Non-CVE vuln. instances
Java	Other	749	46.0	665	84
	Denial of Service	272	17.0	205	67
	Information Disclosure	147	9.0	125	22
	Cryptography	145	9.0	130	15
	Remote Procedure Calls	133	8.0	105	28
Python	Other	70	49.0	69	1
	Information Disclosure	24	17.0	21	3
	Configuration	23	16.0	23	0
	Denial of Service	21	15.0	17	4
	Cross Site Scripting (XSS)	15	10.0	14	1
Ruby	Denial of Service	281	25.0	97	184
	Other	280	25.0	105	175
	Cross Site Scripting (XSS)	274	25.0	182	92
	Information Disclosure	175	16.0	109	66
	SQL Injection	122	11.0	122	0

common across languages ("Denial of Service" and "Information Disclosure"), which is reassuring. However, there are higher percentages of high-severity vulnerabilities in the dependencies of the Java and Python projects. While it is possible that the variance in sample projects' code quality contributes to the difference in severity distribution, Ray et al. [169] reported that the effect size of the association between programming language and code quality is small. This implies that the difference in severity distributions cannot be fully explained by potential code quality difference across the three languages. Overall, the difference suggests that Java and Python developers will benefit more from timely dependency updates.

Finding 3: Most detected dependency vulnerabilities are of medium severity, however, there is noticeable variation in severity distribution across programming languages.

Table 4.9: Distribution of severity of vulnerability instances. Percentages are of all vulnerability instances in the respective programming language group (both CVE and non-CVE).

	Severity	Java		Python		Ruby	
		%	instances	%	instances	%	instances
CVE	Low	0.8	13	1.4	2	4.5	50
	Medium	46.3	757	63.0	90	46.6	518
	High	34.9	571	23.8	34	8.0	89
	Critical	0.8	13	3.5	5	0	0
Non-CVE	Low	1.3	21	0.7	1	0.8	9
	Medium	15.2	249	6.29	9	33.9	377
	High	0.7	12	1.4	1.4	1.6	18
	Critical	0	0	0	0	4.59	51

4.4.1.4 Vulnerable libraries affecting largest number of sampled projects

To see whether the vulnerabilities in the sampled projects originate from a few widely-used libraries, or many libraries that affect few projects each, we investigate the number of projects affected by each vulnerable library. For this analysis we list vulnerabilities discovered at the latest commit of the sampled projects, spanning both vulnerabilities with and without CVE identifier. Afterwards, we identify the library name associated with the vulnerability, and counted the number of repositories affected by each library. For the purpose of this analysis, we do not distinguish specific library version used, and we disregard the specific number of vulnerabilities associated with the library. That is, a library with five detected vulnerabilities and another library with two detected vulnerabilities will both count as one if they are used by one project. The top five result is shown in Table 4.11.

We note that a vulnerable library also affects the security of the entire package ecosystem through other libraries that depend on it (see e.g. Zimmerman et al.'s work on *npm* [227]). Therefore, for context, we also computed the average number of libraries impacted by a library, measured as average out-degree of the transitive closure in the dependency graph of the different package ecosystems (Maven for Java, PyPI for Python, Gem for Ruby). The

Table 4.10: Distribution of severity of top vulnerability types. Percentages are of all vulnerability instances in the respective programming language group (both CVE and non-CVE).

Language	Tag	Type	Percentage of all instances			
			Low	Medium	High	Critical
Java	Other	CVE	0.4	13.6	25.8	0.8
		Non-CVE	0.2	4.5	0.4	0.0
	Denial of Service	CVE	0.0	10.5	2.0	0.0
		Non-CVE	0.4	3.5	0.2	0.0
	Information Disclosure	CVE	0.1	7.2	0.3	0.0
		Non-CVE	0.6	0.7	0.0	0.0
	Cryptography	CVE	0.0	7.5	0.5	0.0
		Non-CVE	0.1	0.9	0.0	0.0
	Remote Procedure Calls	CVE	0.0	3.4	3.1	0.0
		Non-CVE	0.0	1.7	0.0	0.0
Python	Other	CVE	0.0	23.8	21.7	2.8
		Non-CVE	0.0	0.7	0.0	0.0
	Information Disclosure	CVE	0.7	13.3	0.7	0.0
		Non-CVE	0.7	1.4	0.0	0.0
	Configuration	CVE	0.0	4.2	11.9	0.0
		Non-CVE	0.0	0.0	0.0	0.0
	Denial of Service	CVE	0.0	11.2	0.7	0.0
		Non-CVE	0.0	2.8	0.0	0.0
	Cross Site Scripting (XSS)	CVE	0.0	7.7	0.0	2.1
		Non-CVE	0.0	0.7	0.0	0.0
Ruby	Denial of Service	CVE	0.0	6.2	2.5	0.0
		Non-CVE	0.5	10.2	1.3	4.6
	Other	CVE	0.3	6.7	2.4	0.0
		Non-CVE	0.0	9.7	1.4	0.0
	Information Disclosure	CVE	0.2	7.8	1.8	0.0
		Non-CVE	0.1	5.9	0.0	0.0
	Cross Site Scripting (XSS)	CVE	3.8	12.6	0.0	0.0
		Non-CVE	0.0	8.3	0.0	0.0
	SQL Injection	CVE	0.0	9.7	1.3	0.0
		Non-CVE	0.0	0.0	0.0	0.0

counts at the time data collection begins (December 2018) is 81.9 for Java, 10.3 for PyPI, and 62.2 for Gem. This indicates that the higher number of projects affected by the top vulnerable Java and Ruby libraries in the sample set is due to inherent wider impact of average library in Maven and Gem ecosystems. This in turn indicates that improving security in these ecosystems will benefit wider range of projects.

Table 4.11: Top vulnerable libraries by projects affected. Project count includes projects using any version of the specified library.

Language	Library	Projects
Java	Guava: Google Core Libraries for Java	45
	Apache Commons IO	33
	Spring Web	30
	jackson-databind	30
	Apache Commons Collections	28
Python	numpy	23
	PyYAML	9
	Django	7
	requests	4
	Pillow	2
Ruby	rack	59
	nokogiri	51
	loofah	42
	activejob	41
	activerecord	30

Finding 4: Top vulnerable libraries in Java and Ruby affect relatively larger number of projects, in line with wider average impact of libraries in their package ecosystems.

4.4.1.5 Non-CVE dependency vulnerabilities as discovered by Veracode SCA

There are differing views regarding how soon vulnerabilities should be publicly disclosed, taking into account factors such as potential vendor and attacker responses [8]. As a result, there is often a lag between the discovery of a vulnerability by researchers and the inclusion of the vulnerability in the CVE list. Due to this lag, there is a risk that developers may miss some of vulnerabilities in their project dependencies even if they actively monitor and respond to CVE updates. We investigate the extent of such risk by evaluating the average percentage of dependency vulnerabilities in the latest commits of sampled projects that have not been assigned CVE IDs at the time of scan. Table 4.12 shows the average percentage breakdown of CVE and non-CVE

dependency vulnerabilities, along with top tags associated with the non-CVE dependency vulnerabilities. It suggests that while most dependency vulnerabilities discovered in a project are CVE vulnerabilities, developers may still miss a significant percentage of vulnerabilities in their projects' dependencies if they rely on CVE list alone.

Table 4.12: Percentage and top tags for non-CVE vulnerabilities

Language	Percentage of Non-CVE Vulnerability		Top non-CVE Tags
Java	Min	0.0	
	Max	100.0	Other
	Mean	21.9	Denial of Service
	Median	18.2	Cross Site Scripting (XSS)
	Std.dev	24.1	
Python	Min	0.0	
	Max	100.0	Denial of Service
	Mean	5.0	Information Disclosure
	Median	0.0	Buffer Overflows
	Std.dev	17.3	
Ruby	Min	0.0	
	Max	100.0	Denial of Service
	Mean	41.5	Other
	Median	37.5	Cross Site Scripting (XSS)
	Std.dev	28.2	

Finding 5: Relying solely on public vulnerability database may cause developers to miss significant percentage of dependency vulnerabilities.

4.4.1.6 Overall persistence of dependency vulnerabilities

Persistence of dependency vulnerabilities is another aspect that we are interested in, and it is affected by two factors. One factor is how long it takes for the library developers to fix the vulnerability. A number of CVEs affect more than one version of a library, making them persistent despite library updates. An example of this is CVE-2019-17267, which affects the *jackson-databind* library from version 2.0.0 to 2.9.9.4. Another factor is how fast a vulnerable dependency is updated to non-vulnerable version (or removed altogether), since there is often latency in adopting the latest version of libraries [101, 102].

To obtain a general idea regarding the persistence of vulnerabilities across the period of interest, we compute per-project percentage of distinct CVEs (or Veracode SCA artifact IDs in case of non-CVE vulnerabilities) that exist at both the time of the earliest commit and the time of the latest commit in the observation period. Table 4.13 shows the percentage of persistent vulnerabilities for each language grouped by CVSS rating, along with top libraries by the count of persistent CVEs/artifact IDs. We found no clear relationship between survival of vulnerabilities found at first commit and their risk rating. For example, as a group, the vulnerabilities with “Low” rating are least persistent in Java and Python sample projects, but most persistent in Ruby sample projects. This suggests that the high overall persistence is not caused by project owners prioritizing resolution of high-risk vulnerabilities while deferring updates to resolve low-risk vulnerabilities.

Table 4.13: Per-project survival percentages of vulnerabilities present at first commit, grouped by vulnerability risk rating.

Language		CVE				Non-CVE			
		Crit.	High	Med.	Low	Crit.	High	Med.	Low
Java	Projects	13	71	92	13	0	12	75	13
	Min	0.0	0.0	0.0	0.0	N.A.	0.0	0.0	100.0
	Max	100.0	100.0	100.0	100.0	N.A.	100.0	100.0	100.0
	Mean	92.3	81.1	80.7	76.9	N.A.	83.3	85.1	100.0
	Median	100.0	100.0	100.0	100.0	N.A.	100.0	100.0	100.0
	Std.dev	27.7	33.6	34.7	43.9	N.A.	38.9	31.2	0.0
	Top libraries (overall)	jackson-databind, Data Mapper for Jackson, Spring Web, Spring Web MVC, Bouncy Castle Provider							
Python	Projects	3	29	20	3	0	3	7	2
	Min	100.0	100.0	0.0	0.0	N.A.	0.0	0.0	0.0
	Max	100.0	100.0	100.0	100.0	N.A.	100.0	100.0	100.0
	Mean	100.0	100.0	84.2	66.7	N.A.	66.7	71.4	50.0
	Median	100.0	100.0	100.0	100.0	N.A.	100.0	100.0	50.0
	Std.dev	0.0	0.0	34.0	57.7	N.A.	57.7	48.8	70.7
	Top libraries (overall)	Django, numpy, Pillow, PyYAML, requests							
Ruby	Projects	0	38	82	45	51	25	77	11
	Min	N.A.	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Max	N.A.	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Mean	N.A.	61.5	64.4	92.2	88.2	38.3	61.6	54.6
	Median	N.A.	100.0	67.4	100.0	100.0	0.0	66.7	100.0
	Std.dev	N.A.	45.9	36.7	23.7	32.5	47.9	40.0	52.2
	Top libraries (overall)	nokogiri, activerecord, loofah, rack, actionpack							

For context, we also investigate the percentage of dependencies that are not updated or removed since the first commit in the observation period, with the result shown in Table 4.14. In addition, we also conducted survival analysis using Kaplan-Meier method [92] on the library versions found at first commit. As different projects may update their dependencies at different times, for the survival analysis we treat each combination of project and library version as one instance. The result of this analysis is shown in Figure 4.5. Both Table 4.14 and Figure 4.5 show that in many cases, libraries that exist at the beginning of the observation period are not changed by project owners throughout the period. To investigate whether this is due to lack of a newer version of said libraries, we examine per-project percentages of unchanged dependencies for which newer versions already exist. As shown in Table 4.15, in most cases the unchanged dependencies are outdated, yet not replaced. In addition to this, we also investigated vulnerabilities that persist throughout the observation period despite the update of the associated dependencies, with the result shown in Table 4.16. We find that in most of the projects, such vulnerabilities form only a small part of persistent vulnerabilities. Our findings indicate that the persistence of vulnerabilities are caused more by project owners' latency in updating dependencies instead of the vulnerabilities themselves being persistent across library versions.

Table 4.14: Per-project percentages of dependencies in first commit that remains unchanged throughout the observation period. Median and mean commit counts are shown as indicators of sample projects' activity levels.

	Min	Max	Mean	Median	Std.dev	Median commit count	Mean commit count
Java	4.8	100.0	79.6	96.9	27.8	23.0	104.6
Python	20.0	100.0	94.2	100.0	15.7	20.5	76.4
Ruby	2.7	100.0	79.5	100.0	30.8	16.0	56.9

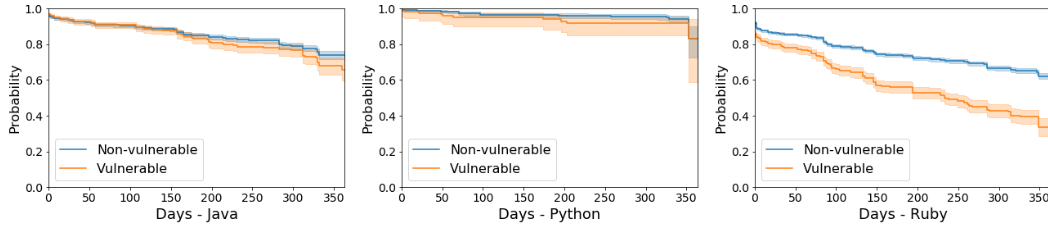

Figure 4.5: Kaplan-Meier curve of vulnerable and non-vulnerable libraries detected at first commit.

Table 4.15: Per-project percentage of unchanged dependencies for which newer version already existed at latest commit. Percentages are of all unchanged dependencies in the same project.

	Min	Max	Mean	Median	Std.dev
Java	0.0	100.0	75.4	80.0	24.0
Python	0.0	100.0	77.6	80.0	21.0
Ruby	0.0	100.0	58.0	57.4	23.4

Table 4.16: Per-project percentage of vulnerabilities that persist despite update of associated dependency. Percentages shown are that of all persistent vulnerabilities in the same project.

	Min	Max	Mean	Median	Std.dev
Java	0.0	100.0	18.4	0	31.9
Python	0.0	100.0	7.4	0	25.8
Ruby	0.0	100.0	36.7	6	43.4

Finding 6: Dependencies are not frequently updated or changed by project owners despite availability of updated libraries, and therefore any vulnerabilities contained will persist.

4.4.1.7 Change of number of dependency vulnerabilities over the period of observation

To examine whether the sampled projects generally become less vulnerable or more over the observation period, we computed the dependency vulnerability counts of each of the 450 projects at their first commits in the observation period (i.e. first commit in November 2017) as well as the latest commits in the period (i.e. latest commit in October 2018). We subsequently apply Wilcoxon signed-rank test on the vulnerability counts at the first and the

latest commits to investigate whether they are significantly different. Since the number of dependencies of a project may also change during the same 1-year period, we also performed the same analysis on dependency counts for comparison. Table 4.17 shows that dependency vulnerability counts tends to decrease despite increase in the number of dependencies in the 1-year period.

Table 4.17: Vulnerability and dependency count changes during observation period. T denotes T statistic of Wilcoxon signed-rank test.

		Java		Python		Ruby	
		CVE	Non-CVE	CVE	Non-CVE	CVE	Non-CVE
Dependency vulnerability counts	T	162.0	25.0	12.0	0.0	0.0	36.0
	p-value	0.000	0.001	0.389	0.046	0.000	0.000
	Min at first commit	0	0	0	0	0	0
	Min at last commit	0	0	0	0	0	0
	Max at first commit	99	19	30	6	46	31
	Max at last commit	98	19	30	6	42	31
	Mean at first commit	9.8	2.1	0.9	0.1	6.6	5.1
	Mean at last commit	9.0	1.9	0.9	0.1	4.4	3.0
	Median at first commit	2.0	1.0	0.0	0.0	1.5	1.0
	Median at last commit	1.0	0.0	0.0	0.0	1.0	1.0
	Std.dev at first commit	15.8	3.5	3.6	0.6	9.9	7.7
	Std.dev at last commit	15.6	3.3	3.6	0.5	7.2	5.2
Dependency counts	T	450.0		124.0		388.0	
	p-value	0.011		0.005		0.000	
	Min at first commit	0		0		0	
	Min at last commit	0		0		0	
	Max at first commit	270		196		241	
	Max at last commit	287		196		356	
	Mean at first commit	35.8		9.6		63.2	
	Mean at last commit	36.0		10.4		66.2	
	Median at first commit	15.5		4		49.5	
	Median at last commit	17		4		50.5	
	Std.dev at first commit	47.6		19.6		54.2	
	Std.dev at last commit	47.5		20.3		61.9	

Finding 7: Dependency vulnerability counts do not increase over the 1-year study period, despite slight increase in number of dependencies.

4.4.1.8 Time required to resolve dependency vulnerabilities

To analyze the time to resolve dependency vulnerabilities, we listed the different dependency vulnerabilities detected in a repository during the observation period. Afterwards, we identify the commit C_1 where the dependency vulnerability is first detected in the repository during the period, as well as the commit C_2 in which the dependency vulnerability is last detected in the same repository. We subsequently identify commit C_3 which is the first commit after C_2 in the repository. For dependency vulnerabilities that already exist at first commit in the period of interest, we use the time of first commit as starting time. We exclude dependency vulnerabilities that still exist at the latest commits. We define the time to fix the dependency vulnerability (i.e. by updating the project's dependency to non-vulnerable version or removing the dependency altogether) as the difference between committer timestamp of C_3 and C_1.

We compute the figure for each repository containing the dependency vulnerability, and subsequently compute the min, max, mean, median, as well as standard deviation of the values. Table 4.18 show the result, broken down into vulnerabilities with and without CVE. We find that on average, the fixed vulnerabilities take 3-5 months to fix. Our finding suggests that dependency vulnerabilities not only tend to be persistent, but even the ones that are resolved take a long time to fix.

Table 4.18: Time taken to fix vulnerabilities in days

Vulnerabilities with CVE					
	Min	Max	Mean	Median	Std.dev
Java	0.0	361.0	145.3	126.0	125.2
Python	0.1	238.7	134.5	174.5	80.7
Ruby	0.0	364.8	98.8	81.3	99.3
Non-CVE Vulnerabilities					
	Min	Max	Mean	Median	Std.dev
Java	0.0	361.0	136.3	150.2	104.3
Python	21.6	228.2	101.3	75.2	79.0
Ruby	0.0	364.8	94.1	69.2	102.1

Finding 8: On average, resolved dependency vulnerabilities take 3-5 months to fix. For vulnerabilities with CVE, average resolution times are 145.3, 134.5, and 98.8 days for our Java, Python, and Ruby datasets respectively. For vulnerabilities without CVE, the average times are 136.3 days for Java, 101.3 days for Python, and 94.1 days for Ruby.

4.4.2 RQ2: What are the relationships between dependency vulnerabilities in a project's open-source dependencies with the attributes of the project and its commits?

4.4.2.1 Project attributes

A popular view regarding open-source software development is reflected in Linus' Law as formulated by Eric Raymond [170]: "Given enough eyeballs, all bugs are shallow". A larger community of developer and reviewers (official testers as well as users) is often expected to improve ability to discover bugs in a software project, including vulnerabilities. This is often used to argue that open source software is more secure [79, 218]. On the other hand, some hold the view that having too many developers can be detrimental (following the notion that "too many cooks spoil the broth"), and there have been studies by Meneely and Williams [130, 131] investigating these opposing views and the extent at which larger number of developers starts to correlate with more vulnerabilities.

Other than the two studies, a number of other works have investigated relationship between the presence of vulnerabilities and various combination of metrics related to software, developer activity, and execution complexity (e.g. [228, 181, 182, 183]), typically with the overall objective of predicting location of vulnerability in a software project's source code. While the focus of

our study is different, considering the view regarding Linus' Law, and another view that higher project complexity and larger project size tend to result in the project being more prone to bugs, we decide to examine whether OSS dependency vulnerability correspond to some project-level metrics: project popularity, complexity, and size. As a proxy for the project's popularity, we use number of commit authors as well as its GitHub stargazers count. As measure of project complexity, we use counts of direct and transitive dependencies of the project, since we are interested strictly in the vulnerabilities resulting from the dependencies instead of the vulnerabilities in the project's own code. Our hypothesis is that larger network of project dependencies will make it more difficult for project maintainers to track and update all dependencies to avoid vulnerable versions.

To investigate the relationship between project attributes and total count of vulnerabilities in its open-source dependencies, we constructed a negative binomial regression model [78]. We chose this regression model because it is more suitable than standard linear regression for non-negative count data [66], and it has also been used in several works in software engineering domain [13, 148, 206, 217]. For this analysis, we use the number of dependency vulnerabilities at the time of latest commit in the observation period as well as the following project attributes:

- **Age**: Project's age, measured as difference between timestamps of project's last commit in the observation period and the first commit in the project repository, in days.
- **Commits**: Total number of commits.
- **Commit authors**: Total number of distinct commit authors.
- **Repository total LOC**: Total LOC in repository excluding test code. Tests are omitted for consistency as Veracode SCA scans ignore test dependencies. Filtering is done by directory, i.e. for Java samples (which

are Maven projects), we exclude *src/test/* which is the typical test location in Maven project structure. For Python and Ruby, we exclude subdirectories named *test/* and *tests/*.

- **Stargazers count**: Number of stars the repository have, as a measure of its popularity.
- **Direct dependencies**: Number of direct dependencies.
- **Transitive dependencies**: Number of transitive dependencies.

We use *statsmodels* [178] implementation of the negative binomial regression, and the results are shown in Table 4.19.

Table 4.19: Negative binomial regression results on project attributes. Shaded cells indicate attributes with statistically significant contribution to dependency vulnerability count.

Variable	Java		Python		Ruby	
	coef	$P > \|z\|$	coef	$P > \|z\|$	coef	$P > \|z\|$
Age	0.000	0.218	0.000	0.596	-0.001	0.007
Commits	0.000	0.922	0.001	0.868	-0.001	0.220
Commit authors	-0.015	0.668	0.072	0.191	-0.030	0.179
LOC	0.000	0.488	0.000	0.254	0.000	0.520
Stargazers	0.000	0.962	0.000	0.079	0.000	0.170
Direct dependencies	0.013	0.109	0.145	0.002	0.031	0.000
Transitive dependencies	0.016	0.000	-0.046	0.006	-0.005	0.103

The results show that the project's age, number of commits, number of developers, popularity, and project size has negligible effect on the dependency vulnerability counts. This suggests that frequent commits, involvement of more developers in a project, and the project's popularity do not translate into better or worse handling of vulnerable dependencies. Possible reasons for the lack of improved handling include lack of awareness regarding the vulnerabilities as well as the presence of dependency constraints that hinder developers from updating project dependencies or switching to a different library (even if there's known vulnerability in the currently-used versions). On the other hand, most direct dependency counts have more statistically significant effects.

Overall, the results suggest that dependency vulnerabilities can likely be managed more effectively through reduction of number of direct dependencies than through recruitment of additional personnel. This reduction can for example be achieved by replacing multiple small libraries with single library that is known to have good security track record.

Finding 9: To mitigate risk from dependency vulnerabilities, managing dependencies will be more effective compared to increasing number of contributors, project activity level, or managing the project's size.

4.4.2.2 Commit attributes

Beside works that focus on predicting vulnerability location using various metrics, a number of works in the field of vulnerability prediction focus on identifying vulnerability-contributing commits [129, 17, 158]. Among other findings, larger changes and developer inexperience have been found to be associated with higher likelihood of a commit introducing vulnerability. Given this, we are also interested in investigating whether experience of developer making the commit or the scale of change caused by a commit correspond with number of dependency vulnerabilities detected after the particular commit. For this analysis, we consider three types of commits:

1. Commits that increase vulnerability count (e.g. due to introduction of vulnerable dependency)
2. Commits that decrease vulnerability count (e.g. due to removal or update of vulnerable dependency)
3. Commits that do not change vulnerability count

The breakdown of the three types of commits for the three programming languages is shown in Table 4.20. As for the commit attributes, we examine the following attributes in this analysis:

- **Developer experience**: We use the number of prior commits in the project as a proxy, since it is not possible to objectively measure and compare actual experience of commit authors directly.
- **Number of affected files**: Total count of files affected by the commit, regardless of operation type (line addition, line deletion etc.).
- **Churn**: Total number of added and deleted lines in the commit.
- **Total LOC of affected files**: Sum of number of lines of code of files affected by the commit, as a measure to distinguish commits affecting small files versus commits affecting large ones.
- **Total Complexity of affected files**: Sum of cyclomatic complexity [127] of files affected by the commit, as a measure to distinguish commits affecting simple files versus commits affecting complex ones.

With the exception of developer experience calculation, we use PyDriller [191] to obtain the metrics. We constructed a logistic regression model using *statsmodels* [178] implementation, and examined the resulting coefficients for the different attributes. The result, shown in Table 4.21, indicates that there is no clear relationship between dependency vulnerabilities with all attributes being examined. A possible explanation is that the dependency changes often occur together with a variety of other changes, such as addition of a large module, a small fix, or deletion of deprecated code. These changes that involve addition or removal of dependencies are diverse in size, for example, a library update from vulnerable versions to non-vulnerable version may only involve changing one line. In addition, as vulnerabilities in dependencies are less visible to developers than issues in their own code, developer experience do not necessarily translate to better handling of security risk from dependencies. In view of this, it seems discouraging large changes on each commit or assigning more experienced developers will not be an effective way to manage security risk

from dependencies. It may be better, for example, for the development team to instead maintain a list of "known-good" libraries that each developer can use as they see fit.

Table 4.20: Counts of the three categories of commits for Java, Python, and Ruby samples

	Java	Python	Ruby
Commits that decreases vulnerability count	69	12	231
Commits that increases vulnerability count	28	8	94
Commits that causes no count change	3936	6125	8901

Table 4.21: Logistic regression results on commit attributes. Shaded cells indicate attributes with statistically significant contribution to dependency vulnerability count.

Attribute	Java		Python		Ruby	
	coef.	$P > \|z\|$	coef.	$P > \|z\|$	coef.	$P > \|z\|$
Vulnerability-increasing commits						
Affected files	0.0069	0.448	0.0291	0.590	-0.0040	0.643
Churn	0.0001	0.026	-0.0010	0.875	-3.004e-06	0.885
LOC	-8.63e-05	0.694	-0.0038	0.177	1.803e-05	0.639
Complexity	-0.0002	0.847	0.0010	0.487	0.0002	0.660
Author experience	-0.0017	0.105	-0.0309	0.123	-0.0002	0.039
Vulnerability-decreasing commits						
Affected files	-0.0051	0.723	0.0237	0.353	0.0012	0.178
Churn	0.0002	0.087	-0.0019	0.644	-2.682e-05	0.042
LOC	-2.865e-06	0.941	-0.0009	0.330	3.784e-05	0.040
Complexity	-0.0010	0.242	0.0005	0.650	-4.729e-05	0.279
Author experience	-0.0010	0.050	-0.0023	0.144	-9.916e-05	0.058

Finding 10: There is no clear relationship between dependency vulnerability count with attributes of the commit including author experience.

4.5 Discussion and Implications

Our results indicate that dependency vulnerability issue affects a wide range of projects, and that such vulnerabilities tend to be persistent, despite overall tendency for the count to decrease.

4.5.1 Implications for library users

Examination of the dataset shows that most libraries used by sampled projects are used transitively, and the number of transitive dependencies is typically much larger than those of direct dependencies. Further, Finding 1 highlights the importance for development teams to perform checks beyond their own code and direct dependencies, and Finding 6 reinforces the need for developers to be vigilant of potential dependency vulnerability beyond those in public database. Finding 7 suggests importance of monitoring and applying updates to project dependencies. Overhead of such effort can be reduced by integrating vulnerability scanning tools or comprehensive software composition analysis tools into the development team's Continuous Integration workflow. In view of typical latency before vulnerabilities become publicly known and additional latency before a fix is available, understanding of common vulnerability types (Finding 2) enables library users to anticipate security risks from such vulnerabilities when designing their software or production environment, for example by applying relevant recommendations from organizations such as OWASP[9].

Finding 9 suggests that there is value in attempting to simplify a project's dependency set to reduce vulnerabilities. Relating our finding to the findings of Abdalkareem et al. [2] regarding prevalent usage of libraries that implement simple tasks but lack tests and introduce many dependencies of their own, one practical step library users can try is to reduce their projects' dependency on such libraries. This can be done, for example, by replacing a group of such libraries with single library that covers the same set of functions and has a good security track record. Beyond this, library users will likely also benefit by selecting a set of libraries that share a common set of dependencies (including the specific version numbers) for their projects.

[9] https://cheatsheetseries.owasp.org/index.html

4.5.2 Implications for library developers

The update latency related to vulnerable dependencies, which contributes to dependency vulnerability persistence (as per Finding 6) and long resolution time (as per Finding 8), suggests that it is important for library developers to make library updates easier, as well as to encourage library users to perform timely update of their projects' dependencies. Given that library users' beliefs regarding potential risks of updating will be strongly affected by personal experience [47], it will be useful to allay library users' concerns about potential risks of update by providing comprehensive tests and documentation, in addition to maintaining good communication with library users.

4.5.3 Implications for researchers

The update latency related to vulnerable dependencies, which result in high persistence of dependency vulnerabilities and long resolution time (Findings 6 and 8), suggests the need for better dependency monitoring and update approaches. One line of work that needs to be explored further is automatic program transformation to allow client code to catch up with the latest updates [56, 107, 200]. Such technique will facilitate smoother dependency update, however, accuracy of existing works is not perfect, and they tend to be limited to particular set of API (e.g. Android APIs). Related to this, our work also demonstrates the value of research into automated techniques to detect breaking changes in library updates, particularly those that are generalizable, as existing works [86, 133, 135] focus on specific language and package ecosystem (Veracode SCA itself supports detection of whether an update is likely to break a build, but supported languages are currently limited to Java, Python, and Ruby).

The findings also demonstrate the value in researching approaches to recommend libraries known to be secure to developers starting new projects, as developers may not readily update or change their project's dependency set af-

terwards, even after the discovery of vulnerabilities. In addition, over lifetime of a project, some of its dependencies may cease to be actively maintained, and those dependencies may subsequently become less secure compared to contemporary alternatives. Detection of such situation and recommendation of alternatives may help project developers keep their work secure. Some tools such as *WhiteSource* and Veracode SCA are able to detect outdated libraries and automatically generate pull request for updates to newer version of the same libraries[10,11]. However, to our knowledge, currently SCA tools do not provide alternative library recommendations based on security track record and update frequency.

Lastly, the prevalence of certain types of dependency vulnerabilities across different languages (e.g. "Denial of Service" and "Information Disclosure") as per Finding 2 indicates potential widespread benefit from research into the resolution or mitigation of such vulnerabilities. It will also be beneficial to conduct future research into common root causes of frequently-discovered vulnerability types, and methods to prevent or detect such issues in library code.

4.6 Threats to Validity

4.6.1 Threats to internal validity

Threat to internal validity stems from limitations related to data and analysis capability of the Veracode SCA tool and its associated platform database. It makes no claim of complete identification of libraries and associated information, and is affected by information in the files it analyzes. We attempt to mitigate this threat by focusing on software projects developed using popular programming languages. Another threat to validity, which affects analyses

[10] https://help.veracode.com/reader/hHHR3gv0wYc2WbCclECf_A/EDLOi6PYdFYDvenrK_0vCQ

[11] https://help.github.com/en/github/managing-security-vulnerabilities/about-security-alerts-for-vulnerable-dependencies

related to correlation between vulnerability and project attributes, originates from the time difference between the latest commit analyzed and the extraction time of the project metadata from GitHub, during which there may be change in attribute's values. Regarding the correlation between the vulnerability and the commit attributes, a threat to internal validity stems from difficulty to accurately measure and compare the experience levels of the commit authors. In this work we used the number of prior commits in the same project as a proxy. The next threat to internal validity, which affects computation of average time needed to fix dependency vulnerabilities, originate from vulnerabilities that have already existed in sample projects since before beginning of the observation period, as well as vulnerabilities that are not yet fixed by the end of the observation period.

4.6.2 Threats to external validity

Generalizability of our findings may be affected by two factors. First, different software projects may use different open-source libraries, which may in turn have different kinds of vulnerabilities and licenses. We attempt to mitigate this threat by performing random selection from *reaper* dataset without regard to project type. Another external threat to validity comes from the fact that the sampled repositories contain projects that have existed for a few years and are still actively developed. While our results indicate no strong correlation between the number of commits in the period of interest and the number of vulnerabilities, there may still be differences between characteristics of the sampled projects with, for example, those of recently started projects that are more likely to use the latest library versions from the beginning.

4.7 Conclusions and Future Work

In this chapter we conducted an empirical study on open-source dependencies of 450 GitHub projects written in three popular programming languages. We scanned the commits made to those projects between November 1, 2017 and October 31, 2018, and identified common vulnerability types, as well as vulnerable libraries that affect the most projects. We also found evidence that number of vulnerabilities associated with open-source dependencies tend to be higher in Java and Ruby projects, indicating opportunity to improve software security by improving open-source libraries, notification of vulnerability discovery, and ease of library update in those languages. Our results indicate that significant percentage of vulnerable dependency issues are persistent, and among the issues that are fixed, the average time taken is about 4-5 months. Related to project and commit attributes, we found that number and experience of contributors, project activity level, and size do not appear to correlate with better handling of vulnerable dependencies. Rather, vulnerability counts correlate more strongly with the number of direct and transitive dependencies. This highlights to library users the importance of managing the number of their projects' dependencies carefully, in addition to performing timely updates.

A potential direction of future work is expansion of the scale of the study to cover projects written in other programming languages supported by Veracode SCA, as well as investigation of commits from longer time period. Beyond this, our future work lies in investigation into associations between dependency vulnerability types as well as the factors that promote or mitigate them. Another element of potential future work related to our study is the identification of characteristics of projects with track record of resolving dependency vulnerabilities quickly and how the characteristics can be emulated in other projects. In addition, we are also interested in investigating the techniques to automatically identify and update vulnerable dependencies in project codebase.

Chapter 5

Understanding Opportunities and Challenges of Geographic Gender-Inclusion in Open Source Software

5.1 Introduction

The gender gap in the software industry is alarming, garnering attention worldwide. IT companies in India reportedly have women concentration in lower career levels [162]. In the United States, women earning computing degrees rose since the mid-1990s, yet they comprise a quarter of computing professionals [49]. An estimate by the European Commission [33] suggests that if more women enter the digital job market, it could create an annual EUR 16 billion GDP boost for the European economy.

Similar investigations in open source software systems show that despite no significant differences between the work practices of men and women [25] and improved team performance in gender-diverse teams [147], women make up less than 10% of core contributors [18]. Further, horizontal and vertical

segregation exist [25].

In open source, explorations on gender diversity are univariate, implicitly assuming that the problem remains the same irrespective of the population and project characteristics. However, in this approach, we are likely to miss local achievements in promoting gender diversity and/or problems unique to others. One factor to consider is the geographical region. A study conducted within the European Union shows a disparity in women's participation in digital economies, with Finland and Sweden scoring the highest while Greece and Italy the lowest [33]. This example suggests that digital and online engagement can shift across geographic regions in addition to genders. Thus, inspiring us to ask how this difference in engagement can manifest in open source, specifically.

Our study presents the largest exploration into gender diversity in open source software projects in different parts of the world. We investigate active and collaboratively developed software projects hosted on GitHub to answer:

RQ1: What are the gender and geographic diversity characteristics of open source software projects on GitHub?

The first question is exploratory, presenting the state-of-the-practice on gender diversity and substantiating the need for exploration. Further, we ask questions to open source software contributors to understand:

RQ2: What factors potentially contribute to the differences in gender and geographic-based developer participation?

Our analysis is based on 21,456 carefully selected software projects on GitHub. We use a sequential mixed-methods approach. First, we quantitatively analyze archived software engineering data of the selected projects to show the state-of-practice of gender diversity worldwide. Next, we survey 1,562 contributors, strategically identified from the selected projects based on gender and geography. We solicit their response in search of factors that can potentially contribute to the differences in developer participation based on gender and geography worldwide.

Our analyses of a decade of development activities on GitHub show small but significant improvements in gender diversity in the last five years. While we celebrate the positive change, it is important to remember that we are far from reaching gender balance. Our study further shows that gender diversity changes over time have not been the same across regions. Some regions such as Eastern Asia and Northern America are (relatively) ahead in gender diversity, while others such as Eastern Europe and Sub-Saharan Africa are still catching up. These differences are also reflected in our investigation of gender and regional related motivations and challenges.

This comprehensive guide of gender-geographic challenges and opportunities can direct future in-depth explorations catering to sub-population needs. For example, one of the opportunity identified here is having a code of conduct. Having a code of conduct can support a two-pronged approach of: 1) allowing lurkers interested in contributing (e.g., including women and other marginalized developers) to feel more comfortable in contributing since they know there are guidelines that can protect them from toxic interactions and 2) signal to developers who are already in the community (e.g., including those that may have been inciting toxic interactions) that there will be repercussions for their actions. Solutions such as these can have a long-term impact to minimize gender gap and uplifting society.

Our contributions are as follows:

1. We present an analysis of the activity and experiences at the intersection of gender and global geographic region.

2. Large-scale global analysis of regional gender diversity spanning 21,456 active GitHub repositories and 70,621 commit authors.

3. Global survey of factors that contribute to the differences in gender and geographic-based developer participation, with 122 respondents across 5 large geographic regions and across genders.

4. A discussion of actionable implications of how to support OSS sub-communities across gender and geographic regions.

5. A publicly available dataset to encourage further investigations.

5.2 Methodology

We used a convergent mixed-methods study approach to answer our research questions [37]. We identified active OSS projects that are likely to be non-toy projects, resolved the gender as well as location of the project contributors, and then distributed a survey to understand their motivations and challenges. The following subsections describe each of these steps in detail.

5.2.1 Identification of Suitable GitHub Repositories

5.2.1.1 Initial Set of Repositories

We chose to use GHTorrent data as it has been widely used in software engineering research, including in works related to diversity (e.g. [211, 147, 197]). Using the latest GHTorrent database dump (1 June 2019), we begin by filtering for repositories that are active, are not toy repositories, and involve collaboration between different developers. We use the following repository criteria:

- The repository has existed for at least 180 days (measured using difference of *updated_at* and *created_at* columns in the GHTorrent data). This is to reduce probability that the project is a "toy" repository (e.g., a user trying a programming tutorial) or a student programming assignment (which usually lasts less than a semester).

- The repository has at least one commit from the beginning of 2018 or later. This is to reduce probability that the project is inactive.

- The repository has at least 10 commits from 4 or more distinct commit authors, none of which are marked 'fake' or 'deleted' GHTorrent.
- The repository is not a fork. We chose not to evaluate forks since we are interested in "core" contributors of a project. In addition, contributions to forks are not always integrated back to the original project and there may also be redundant development between forks and original projects [225].

The above criteria were set to reduce probability of including "toy" projects while avoiding potential elimination of active non-toy projects. Considering rapid growth of GitHub in recent years, we believe the criteria still allows newer OSS projects, for example those created in 2018, to be included in the study.

5.2.1.2 Location Resolution of Commit Authors

We subsequently attempt to resolve the location of the commit authors. As GHTorrent data does not include personal information, we collect additional information through the GitHub API prior to location and gender resolution. For location, resolution is based on value of *country_code* field of the commit author's user information, if available. If the field is empty, location resolution is attempted using other fields in the following order:

1. *location* field. For example, if the commit author specifies "Seattle" as their location, the country assigned will be USA. If they specify "Tokyo", the country assigned will be "Japan".
2. Latitude and longitude (*lat* and *long* fields in GHTorrent data, respectively).
3. *company* field. For example, "Argonne National Lab" or "Puget Sound Regional Council" are considered as evidence that the commit author is

based in the USA. "German National Library" is considered as evidence that the author is based in Germany. Where possible, we attempt to resolve an organization's location using its website and LinkedIn page. In case of multinational organizations, the author's location is considered unresolved unless more specific information such as branch name is provided. For example, "RedHat" will be considered as unresolved location, whereas "RedHat UK" will be considered as evidence that the location is the UK.

4. *email* field. For example, if the author's email address uses an Australian government domain, the country assigned will be Australia.

Considering differences in culture and other factors that may exist within a region (for example, North American countries versus Latin American countries, Western European countries versus Eastern European countries), we also assign three levels of region information to each commit author based on the taxonomy of regions specified by United Nations Statistics Division[1]. For example, if the commit author's resolved location is Kenya, the assigned region information will be "Africa" (region level 1), "Sub-saharan Africa" (region level 2), and "Eastern Africa" (region level 3). Our intention is to facilitate analyses at finer granularity instead of treating a continent (e.g. America, Asia, Europe) as a unit.

5.2.1.3 Gender Resolution of Commit Authors

For the commit authors' gender, resolution is attempted by identifying first name portion of the commit author's name. This is followed by resolution of gender using *genderize.io*[2], which has been reported to have high accuracy [177, 93] and has been used in various studies related to gender representation (e.g. [80, 198, 175]) as well as in the media[3] For this part, titles (e.g.

[1]https://unstats.un.org/unsd/methodology/m49/
[2]http://www.genderize.io
[3]https://genderize.io/use-cases

"Dr.") are ignored, and if the commit author does not use Latin alphabet to specify their name, the name is first converted to Latin alphabet using a combination of CC-CEDICT[4] (for Chinese characters) and Google Translate[5].

As an additional measure to evaluate *genderize.io*'s accuracy, we randomly selected five sample repositories for manual validation. The repositories are associated with a total of 57 contributors from different regions (15 from Americas, 12 from Asia, 18 from Europe, 4 from Oceania, and 8 with unknown region). Each repository is assigned to a person, and each person subsequently attempt manual gender resolution using public information sources (the contributor's GitHub page, LinkedIn page, Twitter profile, etc.). The result is subsequently compared to gender prediction result from *genderize.io*. We find that overall the manual analysis results match *genderize.io*'s results 89.5% of the time, with 100% match on European and Oceanian contributors, 91.7% on Asian contributors, 80% on contributors from Americas. In case of contributors whose location is unresolvable, there is 75% agreement between manual resolution and *genderize.io*'s prediction based on contributors' names.

5.2.1.4 Final Selection of Repositories

Following this, we apply further filtering for repositories for which both gender and location can be resolved for at least 75% of the commit authors. Considering that not all repositories on GitHub are software project repositories [91], we also exclude repositories for which GitHub detects no primary language.In all, after the entire process, 21,456 repositories are shortlisted, with the breakdown of filtering result at various stages shown in Table 5.1. We also extract all commit authors associated with the shortlisted repositories. Tables 5.2 and 5.3 show the statistics of the dataset.

[4]https://cc-cedict.org/wiki/
[5]https://translate.google.com/

Table 5.1: Result of project repository filtering steps.

Filtering step	Count
Initial number of repositories	125,485,095
Repositories with commits newer than January 1, 2018	31,947,039
Repositories that have existed for at least 180 days and are not marked as “deleted	4,393,507
Repositories with at least 10 commits, and are not a fork	2,129,448
Repositories remaining with no commit authors marked “fake” or “deleted”	97,989
Repositories with 75% commit authors having resolvable gender and location	21,456

Table 5.2: Statistics of shortlisted repositories and associated commit authors.

Shortlisted Repositories				
	Min	Max	Mean	Median
No. of Commit Authors	4	109	6.16	5
No. of Commits	22	301692	363.27	170
Creation year	2008	2018	2014.63	2015
Commit Authors of Shortlisted Repositories				
Total commit authors count				70,621
Commit authors with resolvable location				58,498
Commit authors with resolvable gender				65,132
Commit authors with resolvable gender and location				56,866

5.2.1.5 Calculating Gender Diversity of Commit Authors

To measure the gender diversity of commit authors from different regions, we use the Blau diversity index [15] which has also been used in several works in software engineering domain [211, 212, 28]. In simple terms, the index specifies the probability that two randomly-selected members of a group would belong to different categories. It is defined as $1 - \sum_{i \in \{m,f\}} p_i^2$, where p_i^2 are proportion of men and women (“M” and “F”, respectively) among commit authors.

During calculation, we disregard unknown values. For example, if a region is associated with five commit authors, and four of them are identified as men while one is unknown, the gender diversity index will be 0. Similarly, if a set of commit authors from a region comprise two men, two women, and one person

Table 5.3: Commit author region and gender in shortlisted repositories, sorted by Region Level 1.

Region Level 1	Region Level 2	Count	Percentage		
			Man	Woman	Un-known
Africa	Northern Africa	91	91.21	5.49	3.33
Africa	Sub-Saharan Africa	273	92.67	3.66	3.66
Americas	Latin America and the Caribbean	2547	93.29	4.75	1.96
Americas	Northern America	24055	90.27	7.47	2.25
Americas	Others	5	80.00	0.00	20.00
Asia	Central Asia	34	88.24	2.94	8.82
Asia	Eastern Asia	2585	80.46	10.10	9.44
Asia	South-eastern Asia	686	87.90	6.85	5.25
Asia	Southern Asia	1463	91.46	5.47	3.08
Asia	Western Asia	529	93.19	3.40	3.40
Europe	Eastern Europe	3858	94.35	2.90	2.75
Europe	Northern Europe	7541	92.71	5.38	1.91
Europe	Southern Europe	2314	94.77	3.11	2.12
Europe	Western Europe	10637	92.94	3.88	3.18
Oceania	Australia and New Zealand	1870	92.62	5.13	2.25
Oceania	Melanesia	5	80.00	0.00	20.00
Oceania	Polynesia	5	100.00	0.00	0.00
Unknown	Unknown	12123	61.96	6.22	31.82

with unidentified gender, the gender diversity index will be 0.5, which is the maximum value.

To check whether the diversity of commit authors is independent from region, we apply the Chi-squared test to analyze distribution of the two genders across regions, and subsequently computed Cramér's V [36] to measure association strength between gender and region at both region levels. For this analysis, we include commit authors whose location and gender are resolvable (56,866 commit authors comprising 53,426 men and 3,440 women). Exclusion of commit authors with unknown gender is done for consistency with Blau diversity index computation, while exclusion of commit authors with unknown location is done since we are interested in variation between regions worldwide.

Since we note that most projects (70.27%) have a majority region at region level 1, i.e. level 1 region from which more than half commit authors

originate, we also performed a repository-oriented diversity analysis to provide additional perspective. To do this, we first associate a repository to a location based on the most common identified location of the commit authors. For example, if five commit authors contribute to a repository, and their locations are {"Europe", "Americas", "Americas", "Americas", "Asia"}, then the repository will be associated with Americas. Afterwards, we compute the diversity index of each repository. To test statistical significance and effect size of the difference, we first apply Kruskal-Wallis H test on groups of repositories associated with each level 1 regions. We subsequently applied Mann-Whitney U test [123] with Bonferroni correction [3] to compare different pairs of region level 1. Afterwards, we computed Cliff's Delta [32] on level 1 region pairs[6] with statistically significant difference to discover the effect size.

Both the region- and repository-oriented analyses demonstrate low gender diversity worldwide, with similar ordering of regions from least to most diverse. The detailed results are discussed in Section 5.3.1.1.

After we conducted our initial analysis on the commit authors, we also considered following the line of research of Trinkenreich et al. [205] by investigating activities of non-technical contributors. We extracted data of GitHub users who had never authored a commit to the shortlisted sample repositories but had created, changed, or commented on issues and merged pull requests associated with the sample repositories. We excluded user IDs that are marked "fake" and "deleted" in GHTorrent. We found 299,159 users that are not also commit authors. Out of this group, 30.59% has both unresolvable gender and location. Beyond this, 21.56% has unresolvable location although their genders are resolvable, and 9.54% has unresolvable gender although their locations are resolvable. Table 5.4 shows the breakdown of this non-author group by region level 1, along with the Blau index of the users in this group whose gender is resolvable. We note that for members of this group with resolvable gender and

[6] We use https://github.com/neilernst/cliffsDelta implementation for Cliff's Delta test

location, the vast majority is male, and like the case with commit authors, there is low diversity in the various regions studied. However, due to the large percentage of users with unknown gender and/or location among this group, we decided not to analyze this group and to focus our analysis solely on commit authors.

Table 5.4: Diversity and counts of contributors other than commit authors by region level 1. Entries are ordered by non-decreasing Blau index value. Blau index of 0.5 indicate maximum diversity (50% men, 50% women)

Region Level 1	Count				%	Blau index
	M	W	Unknown	Total		
Europe	43873	1402	10303	55578	18.6	0.06
Oceania	3359	121	1022	4502	1.5	0.07
Americas	44859	2430	8909	56198	18.8	0.10
Africa	1432	87	387	1906	0.6	0.11
Asia	15424	1351	7860	24635	8.3	0.15
Unknown	59486	4857	91428	155771	52.2	0.14

5.2.1.6 Examining Correlation between Geographic and Gender Diversity

We are also interested in whether a repository's gender diversity correlates with its geographic diversity. As the Blau index values of repositories' contributor gender and location diversity are not normally distributed (D'Agostino's K^2 test [38] yields p=0.00 for gender diversity index values as well as region diversity index for all levels of regional grouping), we analyze this by computing Spearman's rank correlation test [81] between repositories' gender diversity index values and geographic diversity index values at different regional groupings. We use *SciPy* [214] implementation of these statistical tests, and follow scale of interpretation of ρ used by Camilo et al. [22] ($\pm$ 0.00 - 0.30: Negligible, $\pm$ 0.30 - 0.50: Low, $\pm$ 0.50 - 0.70: Moderate, $\pm$ 0.70 - 0.90: High, and $\pm$ 0.90 - 1.00: Very high). We found no strong correlation between gender and geographic diversity, with the details discussed in Section 5.3.1.2.

5.2.1.7 Examining Gender Diversity Changes over Time

Beyond state of gender diversity based on latest GHTorrent data, we are also interested in how gender diversity changes over time. Considering rapid expansion of GitHub in recent years (it has grown from 10 million repositories by end of 2013 to more than 100 million repositories by November 2018 [68]), we decide to focus our analyses of change on the period from 2014 onwards.

To create a baseline for comparison, we use the GHTorrent commit data to identify a set of GitHub users who have authored at least one commit to shortlisted projects by 2014. We subsequently apply the same approach used for RQ1 to compute diversity index values for different regions in 2014. We then perform Kruskal-Wallis H test to evaluate the statistical significance of the difference in diversity between 2014 and latest state. Afterwards, we calculate the effect size using Cliff's Delta. The result of this analysis, detailed in Section 5.3.1.3, indicates a global increase in gender diversity in OSS projects.

5.2.1.8 Examining Gender Diversity of Older versus Newer Accounts

An additional aspect we are interested in is whether, among commit authors, there is difference in gender balance between older and newer accounts. We investigate this by looking at the account creation years of all commit authors of the shortlisted projects, and compute gender composition for each year between 2014-2018 (the latest year for which GHTorrent has complete data). We find that the percentage of accounts created by women remained low throughout the period, with the breakdown shown in Section 5.3.1.4.

5.2.2 Globally-Distributed Developer Survey

5.2.2.1 Protocol

To understand motivations and challenges faced by developers of different genders in various regions when joining and leaving software projects, we designed and distributed an online survey. The survey comprised three section of questions. The first section solicits the motivation of developers to contribute, frequency of participation, reasons for selecting a particular project, continue participation, as well as barriers and reasons they have abandoned a software project. We build upon previous surveys on barriers and experiences in online programming communities to develop our survey questions in this section [113, 62, 230]. To help participants ground their responses, we asked them to answer the above questions for one of the software projects we identified them from. The second section of our survey included questions about how relevant the gender and region of co-contributors is when selecting a project to contribute to. This section of questions is inspired by how peer parity can encourage participation of people from a shared background or identity [60]. Relating to region, we ask how challenging it is to contribute with people who speak a different language and the usefulness of translation tools to support that interaction. Likewise, we asked about the ease of contributing to projects that have contributors with same gender identity and their advice to encourage women participation in GitHub. Finally, in this section we asked all respondents about what should be done to encourage more women in OSS which is aligned with previous surveys [25, 62]. In asking all respondents, we understand better how to approach interventions that not only serve women, but also those of other marginalized identities across geographic regions. In the third section of our survey, we asked demographic questions about their gender identity and the geographic region they contribute to open source from. All questions were optional and presented as either a Likert scale, multiple-choice, or open response question. The survey was designed to be completed

in approximately 7 minutes.

5.2.2.2 Participants

We identified survey participants from our GHTorrent sample. Our sample comprised all contributors from the selected projects for whom we can infer region, gender, and email address to contact them. The distribution of contributors was skewed towards some regions (e.g., Northern America was over-represented while Micronesia was underrepresented). We observed this skew also in the distribution of men and women across regions.

To gather a representative sample spanning multiple regions, we selected 50 men and 50 women from each region. For over-represented groups such as men and Northern America, we randomly identified 50 participants, while for underrepresented groups (with participants less than 50), we selected all contributors. Overall, we identified 1,562 contributors, of which 1,527 email addresses were valid and did not have an out-of-office reply message. The distribution at region level 2 is shown in Table 5.5, while the total for each region level 1 is shown in Table 5.6.

We received 120 responses (out of 1,527 emails sent; approximately 8% response rate) in three weeks. On reviewing the responses, we manually analyzed the survey responses for anti-patterns (e.g., all responses are empty or have the same value for all questions). We found two responses with all empty values which we discarded from analysis. We did not observe any other patterns in survey responses. We used 118 responses after discarding the two empty responses.

Our survey garnered approximately one response from a woman (total: 23) for every four responses from men (total: 90). Although provided with an option, no participants in our sample identified their gender as non-binary. Our participants have contributed to open source from around the world, including Europe (46), Asia (29), Americas (21), Africa (12), and Oceania (4), with an

Table 5.5: Distribution of surveyed commit authors at region level 2.

Region Level 1	Region Level 2	M	W	Unknown
Africa	Northern Africa	50	3	1
Africa	Sub-Saharan Africa	50	2	2
Americas	Latin America and the Caribbean	50	50	17
Americas	Northern America	50	50	50
Americas	Others	3	0	0
Asia	Central Asia	22	0	1
Asia	Eastern Asia	50	50	50
Asia	South-eastern Asia	50	30	9
Asia	Southern Asia	50	50	15
Asia	Western Asia	50	11	5
Europe	Eastern Europe	50	49	20
Europe	Northern Europe	50	50	20
Europe	Southern Europe	50	39	7
Europe	Western Europe	50	50	50
Oceania	Australia and New Zealand	50	39	10
Oceania	Melanesia	3	0	0
Oceania	Polynesia	4	0	0
Unknown	Unknown	50	50	50

Table 5.6: Distribution of surveyed commit authors at region level 1.

Region Level 1	M	W	Unknown
Africa	100	5	3
Americas	103	100	67
Asia	222	141	80
Europe	200	188	97
Oceania	57	39	10
Unknown	50	50	50

overall distribution shown in Table 5.7. Some participants preferred not to disclose either gender or geographic region; hence the total count in Table 5.7 is lower than the number of responses received.

5.2.2.3 Analysis

We had two types of responses: Likert scale and open-ended. To process Likert scale responses, we transformed an ordinal scale into a nominal scale. For example, a 5-point Likert scale of 'Very important, Important, Neutral,

Table 5.7: Distribution of survey responses based on gender and region.

Region	Men	Women	Total
Europe	35	10	45
Asia	25	4	29
Americas	13	7	20
Africa	11	1	12
Oceania	3	0	3
Total	87	22	109

Less important, and Not at all important' was converted into 'Important' (combining 'Very important' and 'Important' into one), 'Neutral', and 'Not Important' (combining 'Less important' and 'Not at all important' into one). This way it is easier to (statistically) distinguish factors deemed important from not important, in addition to the overall distribution. Similarly, other Likert scale questions were processed.

The transformed nominal scale was fed as input to the Chi-square test to test statistically significant differences in the responses. All tests were conducted in R and reported at $p<0.05$. For data analysis, we analyze aggregates for which we can draw meaningful inferences. Since gendered responses from Oceania are fewer in the count, we remove them from statistical analysis.

For open response survey questions, we conducted a thematic analysis of participant's motivations to contribute, barriers to contribution, and reasons to abandon projects on GitHub. In this analysis, we start with first-cycle descriptive coding [176] (i.e., summarizing the topic of each response as code) on each open-ended response, followed by axial coding (i.e., relating the codes to each other) to connect core experiences respondents had in OSS [23]. This analysis reveals the most important factors to encourage and maintain participation in projects, such as goal alignment and the existence of a welcoming community. Our analysis also identifies differences in how women and men view the importance of factors such as shared gender identity with other project contributors, and how the common motivations to contribute to OSS projects vary among regions. Section 5.3.2 discusses the analysis results in more detail.

5.3 Results

5.3.1 RQ1: What are the Gender and Geographic Diversity Characteristics of OSS Projects on GitHub?

5.3.1.1 Regional Variations

We find that gender diversity of repositories' commit authors are generally low worldwide, as shown in Tables 5.8 and 5.9. Through Chi-squared test, we found relationship between gender and region (p=6.25e-56 at region level 1 and p=1.30e-78 at region level 2) but negligible association strength (Cramér's V result of 0.07 at region level 1 and 0.08 at region level 2).

Table 5.8: Gender diversity (or Blau) index arranged in non-decreasing order by region (level 1). Blau index of 0.5 indicate maximum diversity (50% men, 50% women).

Region	Blau index	Commit authors (% distribution)
Africa	0.08	364 (1%)
Europe	0.08	24350 (34%)
Oceania	0.10	1880 (3%)
Americas	0.14	26607 (38%)
Asia	0.15	5297 (7%)
Unknown	0.17	12123 (17%)

The result of our repository-oriented additional analysis at region level 1, shown in Table 5.10, demonstrates similar ordering from least to most diverse regions. We find that this approach produce overall result that is consistent with result of our previous, region-oriented approach. There is statistically significant difference among regions overall, (p=3.89e-115 in Kruskal Wallis H test). We found three pairs with statistically significant difference in Mann-Whitney U test (Americas versus Europe, Asia versus Oceania, and Asia versus Europe, all of which have $p<0.001$). However, we observe negligible effect sizes on Cliff's Delta test (δ of 0.098 for Americas versus Europe, 0.088 for Asia versus Oceania, and 0.132 for Asia versus Europe).

Table 5.9: Gender diversity index values arranged in non-decreasing order by region (level 2). Blau index of 0.5 indicate maximum diversity (50% men, 50% women).

Region Level 1	Region Level 2	Blau Index	Commit authors
Americas	Others	0.00	5
Oceania	Melanesia	0.00	5
Oceania	Polynesia	0.00	5
Asia	Central Asia	0.06	34
Europe	Eastern Europe	0.06	3858
Europe	Southern Europe	0.06	2314
Africa	Sub-Saharan Africa	0.07	273
Asia	Western Asia	0.07	529
Europe	Western Europe	0.08	10637
Americas	Latin America and the Caribbean	0.09	2547
Europe	Northern Europe	0.10	7541
Oceania	Australia and New Zealand	0.10	1870
Africa	Northern Africa	0.11	91
Asia	Southern Asia	0.11	1463
Asia	South-eastern Asia	0.13	686
Americas	Northern America	0.14	24055
Asia	Eastern Asia	0.20	2585
Unknown	Unknown	0.17	12123

Table 5.10: Gender diversity index values by region level 1, computed by associating project with most frequent contributor location.

Region	Mean	Median	Std. dev.	Min	Max
Europe	0.06	0.00	0.13	0.00	0.50
Africa	0.07	0.00	0.16	0.00	0.50
Oceania	0.08	0.00	0.15	0.00	0.50
Americas	0.09	0.00	0.16	0.00	0.50
Asia	0.11	0.00	0.17	0.00	0.50

Finding 1: Gender diversity is low worldwide, and while there is apparent difference in diversity across regions (with Asia and Americas being highest), statistically the difference is not substantial.

5.3.1.2 Correlation between Geographic and Gender Diversity

The result of our analysis of correlation between geographic and gender diversity, shown in Table 5.11, shows negligible to small negative correlation between gender diversity and geographic diversity. This suggests that project teams that accept contributors from different regions may still be homogeneous in terms of gender, and vice versa, indicating that different approaches are needed to promote each type of diversity.

Table 5.11: Spearman's ρ between repositories' gender diversity and geographic diversity. * indicates p-value <0.001

Regional Grouping	ρ	p-value
Level 1 (e.g. 'Africa')	-0.06	0.00*
Level 2 (e.g. 'Sub-Saharan Africa')	-0.10	0.00*
Level 3 (e.g. 'Eastern Africa')	-0.10	0.00*
Location (e.g. 'Ethiopia')	-0.11	0.00*

Finding 2: There is no strong correlation between gender and geographic diversity.

5.3.1.3 Gender Diversity Changes Over Time

Table 5.12 shows the change in Blau index at region level 2, while Figures 5.1 and 5.2 show the map visualization. We note that there is general trend of improvement, with most regions showing increase in Blau index value, and none show a decrease. We found that the difference between 2014 Blau index values of the various regions and the latest values is statistically significant (p=0.03), and Cliff's Delta calculation indicate large effect size (δ=0.47). However, as shown in Table 5.12, in terms of absolute value, there is still much room for improvement; most regions see an increase in Blau index values of less than 0.10 since 2014, with the exception of Northern Africa, which improved by 0.11.

Table 5.12: Changes in gender diversity of commit authors between 2014 and latest GHTorrent date - region level 2. N.A. indicates regions for which Blau index cannot be computed since there are no users at the time.

Region Level 2	Diversity Index			Users	
	2014	Latest	Change	2014	Latest
Northern Africa	0.00	0.11	0.11	9	91
Sub-Saharan Africa	0.00	0.07	0.07	55	273
Latin America and the Caribbean	0.04	0.09	0.05	563	2547
Northern America	0.09	0.14	0.05	7250	24055
Americas (Others)	N.A.	0.00	N.A.	0	5
Central Asia	0.00	0.06	0.06	6	34
Eastern Asia	0.18	0.20	0.02	772	2585
South-eastern Asia	0.12	0.13	0.01	159	686
Southern Asia	0.08	0.11	0.03	207	1463
Western Asia	0.02	0.07	0.05	113	529
Eastern Europe	0.03	0.06	0.03	962	3658
Northern Europe	0.08	0.10	0.02	2128	7541
Southern Europe	0.05	0.06	0.01	562	2314
Western Europe	0.05	0.08	0.03	2963	10637
Australia and New Zealand	0.08	0.10	0.02	573	1870
Melanesia	0	0.00	0.00	2	5
Polynesia	N.A.	0.00	N.A.	0	5
Unknown	0.11	0.17	0.06	2439	12123

Figure 5.1: Gender diversity at region level 2 as of 2014. Darker shade indicates higher diversity.

Figure 5.2: Gender diversity at region level 2 as per latest data. Darker shade indicates higher diversity.

> **Finding 3:** Globally, the increase in gender diversity in OSS projects is statistically significant with large effect size, however there is still much room for improvement.

5.3.1.4 Gender Diversity of Older versus Newer Accounts

Figure 5.3 shows the breakdown of commit author accounts by creation year and gender. The percentages indicate that the number of GitHub accounts created by women has remained low throughout the period. This suggests a need to encourage participation of women.

> **Finding 4:** Among commit authors with identifiable gender, yearly percentage of account creation by women is around 10%, suggesting that encouragement of participation is still needed.

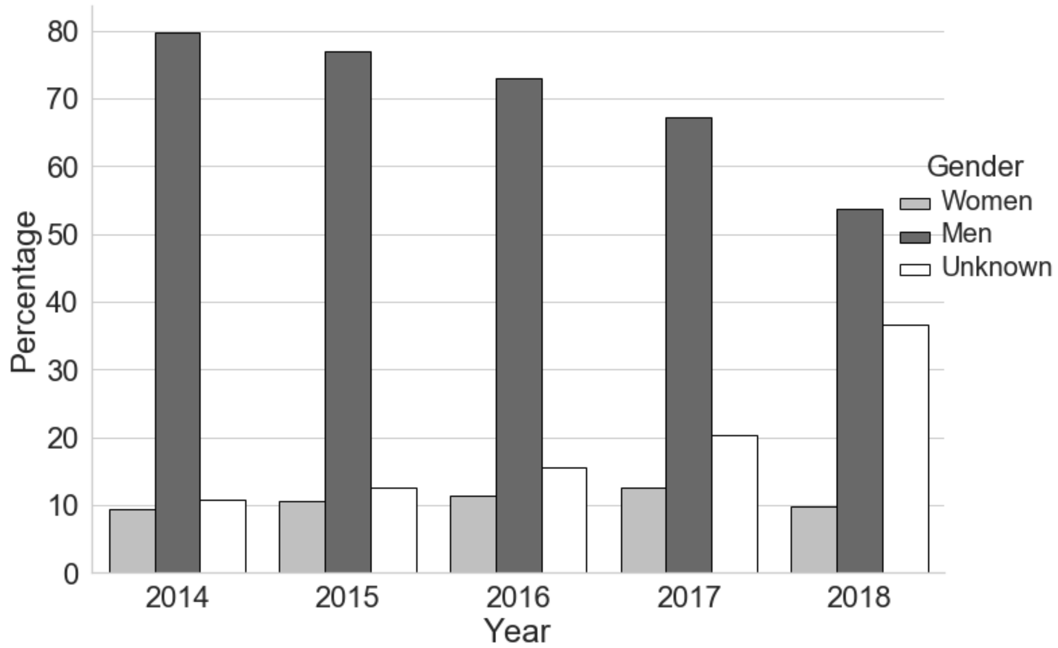

Figure 5.3: Gender percentage of commit authors by account creation year, 2014-2018.

5.3.2 RQ 2: What Factors Potentially Contribute to The Differences in Geographic- and Gender-based Developer Participation?

We received a range of survey responses from participants that include important factors such as the projects impact, how they are motivated by project alignment, and how they have been inhibited by the community culture. In this section we report the results of our analysis which was done at two levels: globally and regionally. Our objective is to obtain both a global view of factors affecting developer participation, as well as view of any region-specific characteristics that can be utilized to promote participation from particular regions.

5.3.2.1 Global Findings

Overall, we find that the majority of survey respondents contribute to GitHub monthly (79), followed by weekly (22), daily (12) and hourly (4) with no differences in contribution pattern across gender and regions.

Project Selection Factors. A majority of developers believe that alignment of project goal to their own is the most important factor for selecting a project. Approximately, 96% of the respondents consider this factor as important while the remaining 4% do not consider it important [χ^2 (1 df) = 86.6, p<0.001]. Other factors deemed important are how welcoming the project is (83% important), how easy it is to join the project (81%), and the opportunity to be a part of how software is built (79%).

Although the majority of participants said they did not select a project because they saw it on social media (94% not important) or that their friends or colleagues contribute to that project (67% not important), few acknowledged how other social dynamics did matter. For example, some participants mentioned how important it was to them that a project "supports social equity (P97)" while providing "up-to-date code for others learning (P125)."

> **Finding 5:** To encourage participation in a project, goal alignment and creation of welcoming community will be more effective than promotion in social media.

Motivations To Contribute. Participants primarily pursued open source software development as their hobby (69 responses), volunteer in the community for free (63), to learn something new (63) or it is their full time job (54) Other less prominent reasons are to get a job (22), meet new people (21), as a part of school or university project (8), and to get paid (6).

From our open responses, participants described their interest in volunteerism as an opportunity to reciprocate what they received from the community in a "socially relevant (P71)" way. One participant goes on to say, "I get so much from the community that I feel where I can I need to give back when I can (P114)."

Motivations to Continue Participation. Once developers have joined a project there are many reasons for developers to continue participation. The factor that is considered most important is interactions with welcoming contributors

(91% of participants consider this important). This is followed by availability of exciting tasks (considered important by 85%) and the global connections they build worldwide (78% important). Low stress level (considered important by 76%) is another common consideration to continue participation.

Finding 5: While developers may participate in a project for variety of reasons, ensuring continued participation requires project owners to maintain welcoming community, ensuring availability of exciting tasks, and minimizing stress to contributors.

Barriers to Contribution. From our analysis, we identified 116 barrier statements referring to reasons contributors have decided not participated in some projects or discontinued contributing from others. From these statements we identified 6 themes.

Lack Of Resources. Participants acknowledged that they had limited resources at their disposal to make significant contributions to a project. These resources included time allocation, the lack of project funding, and challenges balancing time spent on projects for a full time job with projects a hobbyist. One participant goes on to describe his work-hobby balance: "I do not do this as a full time job, I just try to commit meaningful changes that helped me in my own projects (P114)." Another describes their funding challenges: "At times I would like to contribute more but it comes down to a lack of funding to put more hours in. (P112)"

Goal Alignment Shift. As contributors grow in their expertise so do their interests and their professional work. For instance, some participants described how there was a pre-determined end of their "short-lived project (P26)", but also that they, "have abandoned some open source projects because they have been superseded by other projects or because better options for doing the same thing came along (P13)." Participants did not find useful to stay on a project that was no longer a priority.

Inactivity on Projects. Changing project goals often result in projects being abandoned and eventually becoming inactive. Participants described the signs of dying project: "Decrease in the regularity of contributions from project contributors (P70)." This inactivity on the project went beyond who was contributing. Participants also described significant delay in the code review process from maintainers as a barrier: "In general, having no frequent experienced contributors would make me stop contributing because reviews from experienced developers is one of my main motives to contribute (P118)." Contributors are very interested in contributing to projects a as a learning experience, but when the common experience is, "maintainer just stopped reviewing PRs and abandoned the project (P94)," contributors lose value in participating.

Poor Engineering Environment. Factors related to the engineering environment discouraged contributors. Specifically, participants reported being inhibited by the"complex installation process (P71)", "complex code architecture (P70)", "lack of documentation (P71)", and the "lack of a proper roadmap (P110)." Without proper documentation and a clear roadmap of what the north star of a project is contributors will be misguided like P79 who had a challenge finding the best opportunities to help: "On most [projects I'm] not having a clear understanding of what features would be helpful to work on."

Poor Working Environment. Participants disgruntled by their challenges also recalled the toxic work environments some projects can have: "Sure I have stopped contributing to projects when the maintainers are jerks to me or others. Other thing that have curtailed or stopped me from working on a project are racism, misogynous behavior or unprofessional conduct by maintainers (P43)." A few participants went on to to discuss their 1:1 encounters with project leadership: "The big upstream dependency of this project is maintained by a jerk, so I mostly just maintain the project now, rather than actively add new features (P43)." Although these experiences have been described in low frequency, it is important to note that these experiences can influence how

developers decide to contribute like in P43's case.

Unclear Onboarding. The lack of official onboarding documentation processes from maintainers was also discouraging to our participants: "My contribution there was very small, as we did not use it a lot. But I guess this is a good example of the not very well documented project. this is the main obstacle for me when I would like to get involved in some project - not very clear README, missing documentation regarding code discipline for a particular project, not clear rules on how to get involved. That would be for me the main blocker (P98)." When participants reflected on their past experiences with their first project they recalled how challenging it was to join some projects: "The first contact is always the hardest, I mean the totally new newbies always find it intimidating to find and join their first project. (P95)" In short, new contributors to a project have a hard time finding how to get involved.

5.3.2.2 Gender and Regional Related Motivations and Challenges

We found that women developers place high importance on social aspects related to OSS projects as an aspect to consider before participating. Women value selecting a project with friends and colleagues more than men (64% of women participants consider this important, compared to only 25% of men). Beyond this, 37% of women developers believe that shared gender identity with fellow contributors as important, while only 1% of men consider it important. Analysis across regions showed that same gender identity is not at all important for developers from Africa (0%) while it does hold some relevance for other regions: Americas (17%), Europe (11%), and Asia (4%). Beyond the social aspect, we also found that being paid is a greater incentive for women (64% find it important) compared to men (35% find it important).

We also asked participants about what they think can encourage participation among women on GitHub. We found that some men across regions were very dismissive to this question saying, "Ask the women. I'm not stop-

ping them (P9)." On the opposition, we also did find some men suggesting how explicit visibility can inspire others, "There were several women highly qualified for any type of project. But if you need any encouragement, perhaps more women will take the initiative to start new open source projects. Maybe it's contagious (P26)." Likewise, we find that most women were interested in women encouraging other women, but through leadership: "More women reviewers. More women acting directly on the governance of large open source projects (P52)." Additional details about this finding can be found in the Appendix

Finding 6: Shared gender identity, working with friends and colleagues, and being paid is more important for women than men.

5.3.2.3 Regional Variation in Motivations and Challenges

Motivation to Participate in OSS Projects. Table 5.13 shows the developers' motivation to participate in OSS projects, broken down by region. We find that motivation to contribute to OSS as a full-time job is less common outside of Europe and the Americas. In addition, developers from Africa placed a relatively higher importance on networking (i.e., "meeting new people") compared to developers from other regions.

Table 5.13: Motivation of developers to participate in open source software projects across regions. Each cell reports the percentage of developers motivated by the following factors.

	Europe	Asia	Americas	Africa
my full-time job	26.00	11.00	21.00	8.00
my hobby	21.00	28.00	15.00	19.00
volunteer for free	26.00	20.00	17.00	22.00
learn something new	15.00	24.00	25.00	22.00
school/university project	2.00	1.00	8.00	0.00
help get a job	3.00	8.00	8.00	11.00
meet new people	5.00	6.00	6.00	14.00
get paid	2.00	1.00	0.00	3.00

Motivation to Continue Participation in OSS Projects. Table 5.14 shows the developers' motivation to continue their participation in OSS projects, broken down by region. We note that there are regional variations regarding importance of various factors. For example, while exciting and challenging tasks are important for all regions, they are more important for developers from Asia and Africa. On the other hand, connecting with people worldwide is not a big motivation for developers from Europe and Americas to continue participation.

We also found regional differences between what motivates developers to **participate** and what motivates developers to **continue** participation. This difference is in line with Gerosa et al.'s finding [67] regarding shift in motivation of OSS contributors as these contributors gain tenure. For instance, as shown in Table 5.13, the percentage of African developers who participate in OSS as full-time job, to help get a job, or to get paid is relatively small. However, Table 5.14 shows that being paid is an important consideration for African developers to continue participation, much more so than it is for developers from Europe, Asia, and America. This suggests that while African developers may start participating in OSS projects as a hobby, to volunteer, or to learn something new, monetary rewards are important to maintain long-term participation. As another example, while a small percentage of Asian developers stated "meeting new people" as a reason to participate in OSS projects, 89% reported connecting with people worldwide as a reason to continue participation—a percentage similar to developers in Africa (86%).

> **Finding 7:** Some form of funding for participation in OSS projects can be particularly effective to promote continued participation of developers from Africa.

Relevance of Shared Regional and Linguistic Identity. Overall, having contributors from same geographic region in the project is not important for contribution, albeit subtle differences exist across regions. Having contributors

Table 5.14: Reasons to continue participation in open source software projects across regions. Each cell reports the percentage of developers that find the following factors important or not important.

	Europe	Asia	Americas	Africa
Interactions with welcoming contributors				
Important	86	96	94	100
Not important	14	4	6	0
Connects with people worldwide				
Important	67	89	77	86
Not important	32	11	23	14
Exciting tasks				
Important	75	100	77	92
Not important	25	0	23	8
Challenging tasks				
Important	84	100	82	100
Not important	16	0	18	0
Being paid				
Important	34	38	21	71
Not important	66	62	79	29

from the same geographic region is least important for Europe, followed by Americas, Asia and somewhat important for the developers from Africa (see Table 5.15 for details).

We also solicited challenges in working with people who speak a different language, and noticed that while overall differences are not discernible, at regional level, the responses are quite divided. Developers from Europe who happen to see no value in having contributors from same region also do not find it challenging working with developers who speak a different language. Developers from Africa, on the other hand, not only find it relatively more important to have fellow developers from the same region in the project, but also have difficulty in interacting with contributors who speak a language different from theirs. Meanwhile, developers in Asia and America are evenly split in their responses (see Table 5.15 for details). We also found that developers overall hold mixed opinion on the usefulness of translation tools, with no differences across regions. However, there is a difference across genders. We found that 76% of women developers find translation tools helpful, but only

55% of men developers do so.

Table 5.15: Relevance of shared regional identity and language across geographic regions.

	Europe	Asia	Americas	Africa
Contributors from same geographic region				
Important	9	19	15	40
Not important	91	81	85	60
Working with people who speak a different language				
Challenging	26	50	50	80
Not challenging	74	50	50	20

Finding 8: Provision of better translation tools will be particularly helpful to encourage participation of women developers worldwide, as well as participation of developers from Africa.

5.4 Discussion

5.4.1 Summary of Findings

Our result for RQ1 did not show substantial difference across different geographic regions. We note that the set of commit authors with unresolved location has higher apparent Blau index compared to sets from known regions. A factor that contributes to this is the high percentage of users in the set whose gender is also unresolved (31.82%, as shown in Table 5.3). Since the Blau index calculation ignores "Unknown" gender, and majority of commit authors are probably men (based on proportions of commit authors whose gender and location can be resolved), we believe the high percentage of unknowns increases apparent women-to-men ratio in favor of women. This subsequently increases the Blau index of the group with unknown location.

As for the observed diversity improvement during the period analyzed in this work, we believe it is influenced by a combination of factors. Firstly, in recent years there has been increasing interest in promotion of diversity in

computing. This includes efforts by non-profit organizations (such as Girls Who Code[7], Women Who Code [8], NCWIT[9], and ACM-Women [10]), programs targeted at school students [83, 209], initiatives by universities to improve diversity in their own programs [16, 103, 171], as well as efforts by various organizations worldwide to hire more diverse staff. This occurs along the growth of the software industry including in previously underrepresented regions such as Africa [94], with GitHub itself seeing a drastic increase in popularity outside the United States[11]. These factors help attract more diverse talents into computing, including women from underrepresented regions. Nevertheless, as the data shows, there is still much room for improvement.

Related to RQ2, survey responses from our participants encourage us to consider what mechanisms can support contributors from specific regions. In summary, our findings highlight three approaches that should be utilized to better support inclusion across gender and geographic regions. They are:

1. Development of friendlier communities, especially towards newcomers.
2. Highlighting of role models from marginalized communities.
3. Augmentation of existing automated software engineering techniques to incorporate social factors.

5.4.2 Opportunities Ahead

5.4.2.1 Development of Friendlier Communities

There are several ways to encourage development of friendlier, more welcoming communities. Creation and enforcement of codes of conduct are an example of a way to promote a safe environment that can support inclusion [187, 51, 61]. Having a code of conduct can support a two-pronged approach of: 1)

[7] https://girlswhocode.com/
[8] https://www.womenwhocode.com/
[9] https://www.ncwit.org/
[10] https://women.acm.org/
[11] https://github.blog/2018-11-08-100m-repos/

allowing lurkers interested in contributing (e.g., including women and other marginalized developers) to feel more comfortable in contributing since they know there are guidelines that can protect them from toxic interactions and 2) signal to developers who are already in the community (e.g., including those that may have been inciting toxic interactions) that there will be repercussions for their actions. Unfortunately, less than 10% of the top OSS projects actually have one [188]. Participants in our survey also acknowledged that one thing that would encourage inclusion is "Promoting use of and enforcement of code of conduct (P94)." Even fewer projects are transparent about how they enforce these guidelines, if at all.

One approach to enforcing code of conduct usage is rewarding projects that have one. For example, GitHub can offer donation through sponsors program as a reward for projects that have code of conduct. This will provide maintainers with more resources to devote to their role, encourage them to make sure their project is inclusive, and signal to new contributors that a project is safe. Comparatively, this presents a missed opportunity by the projects that have not provided an enforceable code of conduct and thus incentivize those projects to adhere to a new norm. A risk of this approach is the possibility of project maintainers creating token codes of conduct just to satisfy conditions to receive rewards. This approach should therefore be coupled with evaluation of the code of conduct to ensure that it is both meaningful and actually enforced.

Beyond code of conduct, other potential ways to promote development of friendlier communities are usage of social metrics for community self-evaluation and improvement. A example may be drawn from sites that show employer reviews such as GlassDoor[12] and various job search portals. In OSS context, ability to provide and show contributor reviews as well as other metrics such as distribution of contributor tenure can help developers evaluate poten-

[12] https://www.glassdoor.com/

tial projects to join, and also provide an OSS project community a means to evaluate what they have or have not done well and how to improve their community.

Challenge: Many communities currently do not have or enforce code of conduct, and aspiring contributors also can't easily evaluate community quality of a given OSS project.

Opportunity: Improvements can be done by promoting creation and usage of codes of conduct across communities, and to provide set of social metrics to help aspiring contributors evaluate quality of community they consider joining.

5.4.2.2 Mentorship and Highlighting of Role Models

Highlighting of Regional / Women Developers as Role Models. From the responses, contributors from underrepresented OSS regions are not necessarily resentful. Rather, they would like to empower people from their region to take part in the opportunity to be a builder of software that people around the world use [20, 1]. One participant from Sub-Saharan Africa went as far as to state "Open-source software is a solution for Africa to progress as a continent as quickly as possible while spending less money (P23)".

To support and further activate opportunities such as these, we propose a proximity-based mentorship where mentors and mentees are relatively close in region or even close in cultural dimension (e.g., survival vs. self expression [146]). This experience can take advantage of being in the same shared region by conducting guidance through offline interventions [42]. The duality of fostering both the same community online based on a personal offline experience can further support inclusion.

Another approach that can be used is to highlight role models from underrepresented demographics. For example, our survey results indicate that women developers are interested in mechanisms that highlight the contribu-

tion of women. Such mechanisms can be implemented both online and offline. Online mechanisms can be in the form of updates to pages such as GitHub Explore [69] to add sections that highlight rising or top developers from underrepresented communities. For offline implementation of this mechanism, developer communities can for example organize and encourage technical presentations and talks by experienced developers from underrepresented demographics.

Challenge: There is lack of mechanism to highlight contribution of developers from underrepresented demographics.

Opportunity: Mechanisms that highlight developers that are popular globally can be augmented to also highlight top or popular developers from more specific demographics.

5.4.2.3 Diversity Promotion via Automated Software Engineering Tools

Some barriers appear to present opportunities for applying automated software engineering approaches to attract diverse contributors to OSS projects. Existing works [27, 73] highlight the importance of prior social links with existing contributors in developers' decision to join an OSS project, and this can be exploited to promote diversity by augmenting existing approaches with social considerations. We discuss some specific categories of tools in the following paragraphs.

Automated project recommenders can be augmented to take into account social considerations. A small number of recent project recommenders [117, 125] factor in developer's social ties, and GitHub itself takes into account which developers a user "follows" when recommending projects in its GitHub Explore [69] page. However, to promote diversity or participation from particular gender/region, these can be further augmented with additional metrics based on recommendations in the survey responses, for example:

- Metrics related to quality of community. For example, typical tenure of contributors (as a proxy of how much contributors enjoy being in the community), reputation of current contributors, and range of current contributors' experience levels (as a proxy of how welcoming the project is to beginners).

- Number of current contributors known to be from similar region as the developer considering to join the project.

- Diversity of current set of active contributors with known gender and/or location.

Automated documentation improvement can be employed more widely to reduce barriers to contribution. This can include application and enhancement of automated document localization techniques to overcome language barriers and support local languages from regions with large numbers of potential contributors. This may be coupled with application of automated techniques to improve readability, completeness and/or quality of artifacts such as README files [160] and release notes [137]. Usage of automated document generation of source code summary [126] and tracking of outdated API names [114] can further reduce time required from potential contributors. This will be valuable especially in regions where OSS projects are more commonly treated as hobby or volunteer work, since reduced time barrier will enable more people to contribute even without monetary rewards.

Automated developer assignment mechanisms can be updated to distribute exciting / challenging tasks more widely to motivate continued participation. This may be in form of modification to existing automated bug assignment techniques such as [89] and [222], that currently are usually used to speed up resolution process [231] instead of to spreading interesting tasks to team members.

Challenge: Current automated software engineering tools tend to focus on technical aspects and similarity between developers (homophily) when making recommendations.

Opportunity: There's opportunity to augment existing tools to enable selection of target social objectives, such as maintenance of contributor interest (by making more even distribution of challenging tasks) or encouraging participation from certain underrepresented communities.

5.5 Threats to Validity

Construct Validity. Our study has two parts: a large scale data analysis and a survey. During the study design, we made choices that can potentially influence the outcome. Regarding repository selection, the filtering criteria we use still leaves some possibility of including repositories of academic projects that run beyond 6 months, however, we believe that those are also likely to be a more serious endeavor instead of simple programming assignments. Another factor is the accuracy of gender and location resolution. While many factors can cause incorrect gender and location resolution (e.g., incorrect information on GitHub profile, decision to make accounts private), we tried mitigating this threat in two ways. First, we choose a tool that has reportedly reasonable accuracy for multiple regions such as Asia and Eastern Europe [177, 93] and has been used in various studies related to gender representation [80, 198, 175]. Prior to full-scale analysis, we also performed validation by manually checking a subset of the data to increase our confidence in the gender prediction. We also limited our analysis to commit authors, who are more likely to be a code-contributing part of the project team (compared to, for example, issue reporters) and are also more likely to provide information which can be used to resolve their gender and location. Finally, we eliminated projects for whom we could not infer gender and location of at least 75% of commit authors. While it

is also possible to perform additional validation after the survey by comparing self-reported gender and geography in the response to the information inferred from data analysis, we did not do so as we did not ask prior permission from survey participants for such data usage. This is in compliance with the GDPR and broader research ethical considerations.

We also note that the tool that we use (*genderize.io*) is not reflective of a broad gender spectrum. While analysis of non-binary identities is a research challenge that has received increasing research attention [74, 95], we are currently unaware of methods to reliably assess this in software systems at a large scale. Future research should investigate this deeper. As none of our survey respondents identified themselves as non-binary, we believe this limitation of *genderize.io* does not pose a significant threat to the validity of our subsequent analyses.

With respect to our survey, the underrepresentation of women and a broader set of commit authors poses a threat to validity. We attempted to mitigate this by using stratified survey sampling based on gender and location, instead of performing a random sampling of the entire population. For focused survey responses, we asked each participant questions relating to a specific project which we hope provide more concrete response based on the participant's own experience, although there is still some validity risk if the participant has not worked on the project recently.

Internal Validity. Our analysis indicates regional and gender-based differences for open source participants on GitHub. To improve the internal validity of our data analysis, we calculated diversity at different times using two metrics. Our results point in the same direction. Likewise, our survey borrows elements from literature (corroborating with its findings) and builds on it. Using strategic sampling techniques we tried to gather a representative sample to offer a worldwide view.

External Validity. The representatives of our findings is defined by the range

of software projects studied. We selected a wide variety of software projects, nevertheless, we might have systematically missed projects which did not meet our prerequisites (e.g., infer gender and location).

Likewise, due to our methodology and scope of respondents at the intersection of both marginalized genders and underrepresented countries in OSS, we miss the opportunity to provide broad insight into the challenges of having an intersectional identity [167]. Further intersectional methodologies and frameworks should be adopted to explore and amplify the voices of developers in the margins.

5.6 Conclusion and Future Work

In this chapter, we report findings from our large scale empirical study leveraging quantitative data from GitHub and qualitative data for a targeted survey of developers to report on the gender differences across geographies. Our study finds that there is low diversity across regions worldwide, and although there is some variation among regional diversity, the difference is not substantial. Since 2014, there has been small and statistically significant improvement of gender diversity amongst software contributors in North America and South-Eastern Asia but negligible change elsewhere. We observe that among commit authors with identifiable gender, yearly percentage of account creation by women remains low. A qualitative analysis shows that many of the barriers and motivations for contributing converge across different geographic regions ranging from lack of resources, goal alignment shift to poor working environments and unclear on boarding.

There are two underlying themes we hope this study will achieve. The first is quantifying and setting baseline of current state of GitHub regarding intersection of gender and geography. This will help other researchers build on it and quantify changes in coming years. The second is to create awareness

of this problem and hopefully encourage further research by the community towards reducing the gender gap and make software contributions possible by everyone, everywhere. Towards this goal, we are working with people in GitHub and Stack Overflow to help drive some of the concrete observations from our study to alleviate diversity-related issues in the coming years.

Finally, we also believe it will be helpful if researchers from the different parts of the world perform more in-depth study of gender differences in their own regions. We believe that with better understanding of and connections with local developer communities (including developers who are not active on GitHub), local researchers will likely be able to collect more responses. Further, they will also be able to customize their survey to better focus on any region-specific issues they are aware of.

5.7 Dataset Availability

In the interest of encouraging others to replicate and build upon our work, we are sharing our data. The data for this study can be found at: DOI 10.5281/zenodo.4637095

Chapter 6

Conclusion and Future Work

6.1 Summary of Contribution

The increasing popularity of GitHub brings about certain benefits to software engineering community, such as wider range of projects to use and follow. It also reveals several opportunities to bring about widespread improvement to software engineering. Through the works presented in this dissertation's chapters, we examine the quality of software project repositories on GitHub from several aspects. Based on our findings, we also propose ways to create widespread quality improvement of GitHub repositories. Our contributions are summarized below:

- **Categorizing The Content of GitHub README Files**: The first part of our work in Chapter 3 examines the current state of README files in GitHub software project repositories. The findings from the first part add to the body of knowledge regarding the characteristics of GitHub projects, particularly regarding their documentation. These findings can also help project owners to improve the quality of their documentation and make it easier for users and potential contributors to find the information they need. The second part of this chapter describes a README file content classifier we have developed, which achieves F1

score of 0.746. The classifier can be used to automate the task of analyzing the content of a README file and support downstream tasks such as generation of labels for README section headers to indicate information type in the section, or identify information types that are still missing from a README file. To encourage further research, we have made the classifier and our dataset available.

- **Study on Vulnerabilities in Open Source Software Dependencies**: This work, presented in Chapter 4, is an empirical study on open-source dependencies of 450 GitHub projects written in three popular programming languages. Through our analyses of the projects' commits over one year, we identified common vulnerability types in their dependencies. Our subsequent examination of the characteristics of the dependency vulnerabilities reveals that significant percentage of vulnerable dependency issues are persistent, and even those that are fixed took 4-5 months on average to resolve. We also found that vulnerability counts correlate more strongly with the number of dependencies instead of aspects such as project size, project popularity, and contributor experience. Based on our findings, we provide specific recommendations for library users, library developers, as well as researchers to prevent and mitigate such vulnerabilities.

- **Understanding Opportunities and Challenges of Geographic Gender-Inclusion in Open Source Software**: Our third work, discussed in detail in Chapter 5, presents result of large-scale analysis of regional gender diversity spanning 21,456 active GitHub repositories and 70,621 commit authors. In addition, through strategically-targeted global survey and qualitative analysis of the result, in this work we also present motivation and barriers that affect gender and geographic-based developer participation in various regions. Further, based on the findings, we also present a set of recommendations on possible approaches to en-

courage contribution from different genders and geographic regions. To encourage further investigations, we have also made our dataset publicly available.

6.2 Future Directions

The huge number of repositories on GitHub and its rapid growth of popularity worldwide introduces several challenges including identifying well-documented projects, addressing security concerns, and identifying factors that may motivate or hinder good potential contributors from different genders and regions. The works done as part of this dissertation try to address these challenges, and can subsequently be expanded in several directions.

The README file content classifier described in Chapter 3 can be improved to yield higher precision and recall. Beyond this, it can also be used as part of a larger system to enable structured approach to searching and navigating GitHub README files. An example of this would be a search interface to find GitHub repositories that meet certain standards of README quality (e.g. "contains information on basic usage and project status"). Combined with research on other GitHub documentation artifacts such as license information [213] or contribution guidelines [52], this work can also be used to develop approaches to reorganize README files or automatically add relevant content using other artifacts as source of information.

With regard to the work on dependency vulnerability described in Chapter 4, a potential future direction is to conduct a larger-scale study involving more projects and programming languages, which may lead to more generalizable findings regarding common characteristics of, and differences between, dependency vulnerabilities in different types of programming languages. Beyond this, our work can contribute towards research into library developers and users' behaviour related to dependency vulnerabilities. Yet another potential

research direction related to this work is investigation into software projects whose owners have good track record of managing dependency vulnerabilities, and identification of “best practices” that can be emulated by owners of other projects.

Finally, our work on geographic gender-inclusion described in Chapter 5 can form a basis for more in-depth studies of gender diversity in different parts of the world. We hope our work can motivate researchers from different regions to conduct such studies, utilizing their deeper understanding of local context to examine region-specific issues as well as potential solutions. Our work can also be used to inform future research efforts related to how OSS project teams can attract and retain contributors from diverse genders and regions, be it through technical approach (such as incorporation of diversity-promoting features to existing recommendation techniques) or through social approach (such as defining and implementing codes of conduct).

Chapter 7

List of Publications

The works performed as part of this dissertation have resulted in the following publications:

1. **Prana, G. A. A.**, Treude, C., Thung, F., Atapattu, T., and Lo, D. (2019). "Categorizing The Content of Github Readme Files". *Empirical Software Engineering, 24(3)*, 1296-1327.

2. **Prana, G. A. A.**, Sharma, A., Shar, L. K., Foo, D., Santosa, A. E., Sharma, A., and Lo, D. (2021). "Out of Sight, Out of Mind? How Vulnerable Dependencies Affect Open-Source Projects". *Empirical Software Engineering*, 26(4), 1-34.

3. **Prana, G. A. A.**, Ford, D., Rastogi, A., Lo, D., Purandare, R., and Nagappan, N. (2021). "Including Everyone, Everywhere: Understanding Opportunities and Challenges of Geographic Gender-Inclusion in OSS". *IEEE Transactions on Software Engineering.*

Bibliography

[1] Adewale Abati. Made in nigeria, 2017. Retrieved March 5, 2020 from `https://www.madeinnigeria.dev/`.

[2] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 385–395. ACM, 2017.

[3] Hervé Abdi. Bonferroni and šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics*, 3:103–107, 2007.

[4] Surafel Lemma Abebe, Nasir Ali, and Ahmed E. Hassan. An empirical study of software release notes. *Empirical Software Engineering*, 21(3):1107–1142, 2016.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the International Conference on Management of Data*, pages 207–216, New York, NY, USA, 1993. ACM.

[6] Shaosong Ou Alexander Hars. Working for free? motivations for participating in open-source projects. *International journal of electronic commerce*, 6(3):25–39, 2002.

[7] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement?: A text-based

approach to classify change requests. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pages 23:304–23:318, New York, NY, USA, 2008. ACM.

[8] Ashish Arora and Rahul Telang. Economics of software vulnerability disclosure. *IEEE security & privacy*, 3(1):20–25, 2005.

[9] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K. Roy, and Kevin A. Schneider. Answering questions about unanswered questions of Stack Overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 97–100, Piscataway, NJ, USA, 2013. IEEE Press.

[10] Ann Barcomb, Klaas-Jan Stol, Dirk Riehle, and Brian Fitzgerald. Why do episodic volunteers stay in floss communities? In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 948–959. IEEE, 2019.

[11] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in software engineering: A systematic literature review. *Information and software technology*, 50(9-10):860–878, 2008.

[12] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *IEEE Software*, 30(1):52–66, 2013.

[13] Robert M. Bell, Thomas J. Ostrand, and Elaine J. Weyuker. The limited impact of individual developer data on software defect prediction. *Empirical Software Engineering*, 18(3):478–505, Jun 2013.

[14] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* O'Reilly Media, Inc., 2009.

[15] Peter Michael Blau. *Inequality and heterogeneity: A primitive theory of social structure*, volume 7. Free Press New York, 1977.

[16] Valeria Borsotti. Sigsoft distinguished paper-barriers to gender diversity in software development education: Actionable insights from a danish case study. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 146–152. IEEE, 2018.

[17] Amiangshu Bosu, Jeffrey C Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. Identifying the characteristics of vulnerable code changes: An empirical study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 257–268, 2014.

[18] Amiangshu Bosu and Kazi Zakia Sultana. Diversity and inclusion in open source software (oss) projects: Where do we stand? In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019.

[19] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678, 2017.

[20] Zev Brodsky. 7 chinese open source projects you should know about, 2018. Retrieved March 5, 2020 from `https://resources.whitesourcesoftware.com/blog-whitesource/7-chinese-open-source-projects-you-should-know-about`.

[21] Mircea Cadariu, Eric Bouwers, Joost Visser, and Arie van Deursen. Tracking known security vulnerabilities in proprietary software systems. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 516–519. IEEE, 2015.

[22] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. Do bugs foreshadow vulnerabilities?: a study of the chromium project. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 269–279. IEEE Press, 2015.

[23] John L. Campbell, Charles Quincy, Jordan Osserman, and Ove K. Pedersen. Coding in-depth semistructured interviews. *Sociological Methods & Research*, 42(3):294–320, aug 2013.

[24] Eduardo Cunha Campos and Marcelo de Almeida Maia. Automatic categorization of questions from Q&A sites. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 641–643, New York, NY, USA, 2014. ACM.

[25] Edna Dias Canedo, Rodrigo Bonifácio, Márcio Vinicius Okimoto, Alexander Serebrenik, Gustavo Pinto, and Eduardo Monteiro. Work practices and perceptions from women core developers in oss communities. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2020.

[26] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Multi-objective cross-project defect prediction. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 252–261. IEEE, 2013.

[27] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: the role of prior social links and language experience. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 817–828, 2015.

[28] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. Gender diversity and women in software teams: How do they affect community smells? In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 11–20. IEEE, 2019.

[29] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. Detecting missing information in bug descriptions. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*, pages 396–407. ACM, 2017.

[30] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[31] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, pages 767–778, New York, NY, USA, 2014. ACM.

[32] Norman Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[33] European Commission. Increase in gender gap in the digital sector - study on women in the digital age. `http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=50224`, 2018. study reference: SMART 2016/0025.

[34] Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

[35] Denzil Correa and Ashish Sureka. Chaff from the wheat: Characterization and modeling of deleted questions on Stack Overflow. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 631–642, New York, NY, USA, 2014. ACM.

[36] Harald Cramer. Mathematical methods of statistics (princeton: Princeton universitypress, 1946). *CramérMathematical Methods of Statistics1946*, 1946.

[37] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE publications, 2018.

[38] Ralph B D'agostino, Albert Belanger, and Ralph B D'Agostino Jr. A suggestion for using powerful and informative tests of normality. *The American Statistician*, 44(4):316–321, 1990.

[39] Stanislav Dashevskyi, Achim D Brucker, and Fabio Massacci. On the security cost of using a free and open source component in a proprietary product. In *International Symposium on Engineering Secure Software and Systems*, pages 190–206. Springer, 2016.

[40] Steven Davies and Marc Roper. What's in a bug report? In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, page 26. ACM, 2014.

[41] Lucas B. L. de Souza, Eduardo C. Campos, and Marcelo de A. Maia. Ranking crowd knowledge to assist software development. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 72–82, New York, NY, USA, 2014. ACM.

[42] Debian. Debian women mentoring program, 2019. Retrieved March 5, 2020 from `https://www.debian.org/women/mentoring`.

[43] Alexandre Decan, Tom Mens, Maëlick Claes, and Philippe Grosjean. When GitHub meets CRAN: An analysis of inter-repository package dependency problems. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pages 493–504, Piscataway, NJ, USA, 2016. IEEE.

[44] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 181–191. IEEE, 2018.

[45] Alexandre Decan, Tom Mens, and Philippe Grosjean. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, 24(1):381–416, 2019.

[46] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200. ACM, 2017.

[47] Premkumar Devanbu, Thomas Zimmermann, and Christian Bird. Belief & evidence in empirical software engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 108–119. IEEE, 2016.

[48] Wei Ding, Peng Liang, Antony Tang, and Hans Van Vliet. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology*, 56(6):545–567, 2014.

[49] W DuBow and AS Pruitt. Ncwit scorecard: The status of women in technology. *NCWIT, Boulder, CO*, 2018.

[50] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Pax-

son, Michael Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. ACM, 2014.

[51] Coraline Ada Ehmke. Contributor covenant, 2014. Retrieved March 5, 2020 from `https://www.contributor-covenant.org/`.

[52] Omar Elazhary, Margaret-Anne Storey, Neil Ernst, and Andy Zaidman. Do as i do, not as i say: Do contribution guidelines match the github contribution process? In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 286–290. IEEE.

[53] Ali Erdem, W. Lewis Johnson, and Stacy Marsella. Task oriented software understanding. In *Proceedings of the 13th International Conference on Automated Software Engineering*, pages 230–239, Washington, DC, USA, 1998. IEEE Computer Society.

[54] Katalin Erdös and Harry M. Sneed. Partial comprehension of complex programs (enough to perform maintenance). In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 98–105, Washington, DC, USA, 1998. IEEE Computer Society.

[55] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[56] Mattia Fazzini, Qi Xin, and Alessandro Orso. Automated api-usage update for android apps. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 204–215, 2019.

[57] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project.* O'Reilly Media, Inc., Sebastopol, CA, USA, 2005.

[58] Darius Foo, Ming Yi Ang, Jason Yeo, and Asankhaya Sharma. Sgl: A domain-specific language for large-scale analysis of open-source code. In

2018 IEEE Cybersecurity Development (SecDev), pages 61–68. IEEE, 2018.

[59] Darius Foo, Hendy Chua, Jason Yeo, Ming Yi Ang, and Asankhaya Sharma. Efficient static checking of library updates. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 791–796. ACM, 2018.

[60] Denae Ford, Alisse Harkins, and Chris Parnin. Someone like me: How does peer parity influence participation of women on stack overflow? In *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC)*, pages 239–243. IEEE, 2017.

[61] Denae Ford, Reed Milewicz, and Alexander Serebrenik. How remote work can foster a more inclusive environment for transgender developers. In *2019 IEEE/ACM 2nd International Workshop on Gender Equality in Software Engineering (GE)*, pages 9–12. IEEE, 2019.

[62] Denae Ford, Justin Smith, Philip J. Guo, and Chris Parnin. Paradise unplugged: Identifying barriers for female participation on stack overflow. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 846–857, New York, NY, USA, 2016. Association for Computing Machinery.

[63] A César C França, Tatiana B Gouveia, Pedro CF Santos, Celio A Santana, and Fabio QB da Silva. Motivation in software engineering: A systematic review update. In *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, pages 154–163. IET, 2011.

[64] César França, Fabio QB Da Silva, and Helen Sharp. Motivation and satisfaction of software engineers. *IEEE Transactions on Software Engineering*, 46(2):118–140, 2018.

[65] Thomas Fritz and Gail C. Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of the International Conference on Software Engineering - Volume 1*, pages 175–184, New York, NY, USA, 2010. ACM.

[66] William Gardner, Edward P Mulvey, and Esther C Shaw. Regression analyses of counts and rates: Poisson, overdispersed poisson, and negative binomial models. *Psychological bulletin*, 118(3):392, 1995.

[67] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. *arXiv preprint arXiv:2101.10291*, 2021.

[68] GitHub. Github milestones: A timeline of significant moments in github's history, 2020. Retrieved May 28, 2020 from `https://github.com/about/milestones`.

[69] GitHub. Explore github, 2021. Retrieved March 19, 2021 from `https://github.com/explore`.

[70] Gillian J. Greene and Bernd Fischer. Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 804–809, New York, NY, USA, 2016. ACM.

[71] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *Proceedings of the 30th International Conference on Automated*

Software Engineering, pages 771–776, Piscataway, NJ, USA, 2015. IEEE Press.

[72] Nicole Haenni, Mircea Lungu, Niko Schwarz, and Oscar Nierstrasz. Categorizing developer information needs in software ecosystems. In *Proceedings of the International Workshop on Ecosystem Architectures*, pages 1–5, New York, NY, USA, 2013. ACM.

[73] Jungpil Hahn, Jae Yun Moon, and Chen Zhang. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391, 2008.

[74] Foad Hamidi, Morgan Klaus Scheuerman, and Stacy M Branham. Gender recognition or gender reductionism? the social implications of embedded gender recognition systems. In *Proceedings of the 2018 chi conference on human factors in computing systems*, pages 1–13, 2018.

[75] Foyzul Hassan and Xiaoyin Wang. Mining readme files to support automatic building of Java projects in software repositories: Poster. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 277–279, Piscataway, NJ, USA, 2017. IEEE Press.

[76] Claudia Hauff and Georgios Gousios. Matching GitHub developer profiles to job advertisements. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 362–366, Piscataway, NJ, USA, 2015. IEEE Press.

[77] James D. Herbsleb and Eiji Kuwana. Preserving knowledge in design projects: What designers need to know. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, pages 7–14, New York, NY, USA, 1993. ACM.

[78] Joseph M Hilbe. *Negative binomial regression.* Cambridge University Press, 2011.

[79] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Communications of the ACM*, 50(1):79–83, 2007.

[80] Luke Holman, Devi Stuart-Fox, and Cindy E Hauser. The gender gap in science: How long until women are equally represented? *PLoS biology*, 16(4):e2004956, 2018.

[81] Will G Hopkins. *A new view of statistics.* Will G. Hopkins, 1997.

[82] Daqing Hou, Kenny Wong, and H. James Hoover. What can programmer questions tell us about frameworks? In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 87–96, Piscataway, NJ, USA, 2005. IEEE.

[83] Nwannediya Ada Ibe, Rebecca Howsmon, Lauren Penney, Nathaniel Granor, Leigh Ann DeLyser, and Kevin Wang. Reflections of a diversity, equity, and inclusion working group based on data from a national cs education program. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 711–716, 2018.

[84] Nasif Imtiaz, Justin Middleton, Joymallya Chakraborty, Neill Robson, Gina Bai, and Emerson Murphy-Hill. Investigating the effects of gender bias on github. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 700–711. IEEE, 2019.

[85] Sae Young Jeong, Yingyu Xie, Jack Beaton, Brad A. Myers, Jeff Stylos, Ralf Ehret, Jan Karstens, Arkin Efeoglu, and Daniela K. Busse. Improving documentation for eSOA APIs through user studies. In *Proceedings of the 2nd International Symposium on End-User Development*, pages 86–105, Berlin, Heidelberg, 2009. Springer-Verlag.

[86] Kamil Jezek and Jens Dietrich. Api evolution and compatibility: A data corpus and tool evaluation. *Journal of Object Technology*, 16(4):2–1, 2017.

[87] Matthieu Jimenez, Mike Papadakis, and Yves Le Traon. Vulnerability prediction models: A case study on the linux kernel. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 1–10. IEEE, 2016.

[88] W. Lewis Johnson and Ali Erdem. Interactive explanation of software systems. *Automated Software Engineering*, 4(1):53–75, 1997.

[89] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21(4):1533–1578, 2016.

[90] 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*, pages 92–101, New York, NY, USA, 2014. ACM.

[91] 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*, pages 92–101. ACM, 2014.

[92] Edward L Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[93] Fariba Karimi, Claudia Wagner, Florian Lemmerich, Mohsen Jadidi, and Markus Strohmaier. Inferring gender from names on the web: A compar-

ative evaluation of gender detection methods. In *Proceedings of the 25th International conference companion on World Wide Web*, pages 53–54, 2016.

[94] Tim Kelly and Rachel Firestone. How tech hubs are helping to drive economic growth in africa. 2016.

[95] Os Keyes. The misgendering machines: Trans/hci implications of automatic gender recognition. *Proceedings of the ACM on human-computer interaction*, 2(CSCW):1–22, 2018.

[96] Sunghun Kim, E James Whitehead Jr, and Yi Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196, 2008.

[97] Douglas Kirk, Marc Roper, and Murray Wood. Identifying and addressing problems in object-oriented framework reuse. *Empirical Software Engineering*, 12(3):243–274, 2007.

[98] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.

[99] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1028–1038, 2016.

[100] Victor Kuechler, Claire Gilbertson, and Carlos Jensen. Gender differences in early free and open source software joining process. In *IFIP International Conference on Open Source Systems*, pages 78–93. Springer, 2012.

[101] Raula Gaikovina Kula, Daniel M. German, Takashi Ishio, and Katsuro Inoue. Trusting a library: A study of the latency to adopt the latest

maven release. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, page 520–524. IEEE, Mar 2015.

[102] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417, 2018.

[103] Anagha Kulkarni, Ilmi Yoon, Pleuni S Pennings, Kazunori Okada, and Carmen Domingo. Promoting diversity in computing. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 236–241, 2018.

[104] Niraj Kumar and Premkumar T. Devanbu. Ontocat: Automatically categorizing knowledge in API documentation. *CoRR*, abs/1607.07602:preprint, 2016.

[105] Zijad Kurtanović and Walid Maalej. Mining user rationale from software reviews. In *Proceedings of the 25th International Requirements Engineering Conference*, pages 61–70, Piscataway, NJ, USA, 2017. IEEE.

[106] Karim R Lakhani and Robert G Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. 2003.

[107] Maxime Lamothe, Weiyi Shang, and Tse-Hsun Chen. A4: Automatically assisting android api migrations using code examples. *arXiv preprint arXiv:1812.04894*, 2018.

[108] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[109] Thomas D. LaToza and Brad A. Myers. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[110] Tobias Lauinger, Abdelberi Chaabane, and Christo B. Wilson. Thou shalt not depend on me. *Commun. ACM*, 61(6):41–47, May 2018.

[111] Tien-Duy B Le, Ferdian Thung, and David Lo. Theory and practice, do they match? a case with spectrum-based fault localization. In *2013 IEEE International Conference on Software Maintenance*, pages 380–383. IEEE, 2013.

[112] Amanda Lee and Jeffrey C Carver. Floss participants' perceptions about gender and inclusiveness: a survey. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 677–687. IEEE, 2019.

[113] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: A survey. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 187–197, May 2017.

[114] Seonah Lee, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang. Automatic detection and update suggestion for outdated api names in documentation. *IEEE Transactions on Software Engineering*, 2019.

[115] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[116] Jingyue Li, Reidar Conradi, Christian Bunse, Marco Torchiano, Odd Petter N Slyngstad, and Maurizio Morisio. Development with off-the-shelf components: 10 facts. *IEEE software*, 26(2):80–87, 2009.

[117] Chao Liu, Dan Yang, Xiaohong Zhang, Baishakhi Ray, and Md Masudur Rahman. Recommending github projects for developer onboarding. *IEEE Access*, 6:52082–52094, 2018.

[118] David Lo and Xin Xia. Fusion fault localizers. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 127–138, 2014.

[119] Oscar Luaces, Jorge Díez, José Barranquero, Juan José del Coz, and Antonio Bahamonde. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 1(4):303–313, 2012.

[120] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. On the automatic classification of app reviews. *Requirements Engineering*, 21(3):311–331, 2016.

[121] Walid Maalej and Martin P. Robillard. Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering*, 39(9):1264–1282, 2013.

[122] Anas Mahmoud and Grant Williams. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, 21(3):357–381, 2016.

[123] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[124] Umme Ayda Mannan, Iftekhar Ahmed, Rana Abdullah M Almurshed, Danny Dig, and Carlos Jensen. Understanding code smells in android applications. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 225–236. IEEE, 2016.

[125] Tadej Matek and Svit Timej Zebec. Github open source project recommendation system. *arXiv preprint arXiv:1602.02594*, 2016.

[126] Paul W McBurney and Collin McMillan. Automatic documentation generation via source code summarization of method context. In *Proceedings*

of the 22nd International Conference on Program Comprehension, pages 279–290, 2014.

[127] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[128] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. Open source barriers to entry, revisited: A sociotechnical perspective. In *Proceedings of the 40th International Conference on Software Engineering*, pages 1004–1015, 2018.

[129] Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodriguez Tejeda, Matthew Mokary, and Brian Spates. When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 65–74. IEEE, 2013.

[130] Andrew Meneely and Laurie Williams. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 453–462, 2009.

[131] Andrew Meneely and Laurie Williams. Strengthening the empirical analysis of the relationship between linus' law and software security. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.

[132] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango-Argoty. Secure coding practices in java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 372–383. IEEE, 2018.

[133] Gianluca Mezzetti, Anders Møller, and Martin Toldam Torp. Type regression testing to detect breaking changes in node. js libraries. In *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[134] Matthew B. Miles and A. Michael Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE publications, 1994.

[135] Anders Møller and Martin Toldam Torp. Model-based testing of breaking changes in node. js libraries. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 409–419, 2019.

[136] Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. What should developers be aware of? an empirical study on the directives of api documentation. *Empirical Software Engineering*, 17(6):703–737, 2012.

[137] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Automatic generation of release notes. In *Proceedings of the International Symposium on Foundations of Software Engineering*, pages 484–495. ACM, 2014.

[138] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[139] John Mylopoulos, Alex Borgida, and Eric Yu. Representing software engineering knowledge. *Automated Software Engineering*, 4(3):291–317, 1997.

[140] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 284–292. ACM, 2005.

[141] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391. IEEE Press, 2013.

[142] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming Q&A in StackOverflow. In *Proceedings of the International Conference on Software Maintenance*, pages 25–34, Washington, DC, USA, 2012. IEEE Computer Society.

[143] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *ACM Conference on computer and communications security*, pages 529–540. Citeseer, 2007.

[144] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. Focus: A recommender system for mining api function calls and usage patterns. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1050–1060. IEEE, 2019.

[145] Janet Nykaza, Rhonda Messinger, Fran Boehme, Cherie L. Norman, Matthew Mace, and Manuel Gordon. What programmers really want: Results of a needs assessment for sdk documentation. In *Proceedings of the 20th Annual International Conference on Computer Documentation*, pages 133–141, New York, NY, USA, 2002. ACM.

[146] Nigini Oliveira, Nazareno Andrade, and Katharina Reinecke. Participation differences in q&a sites across countries: Opportunities for cultural adaptation. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, NordiCHI '16, New York, NY, USA, 2016. Association for Computing Machinery.

[147] Marco Ortu, Giuseppe Destefanis, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. How diverse is your team? investigating gender and nationality diversity in github teams. *Journal of Software Engineering Research and Development*, 5(1):1–18, 2017.

[148] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Programmer-based fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10*, page 1. ACM Press, 2010.

[149] Andy Ozment and Stuart E Schechter. Milk or wine: does software security improve with age? In *USENIX Security Symposium*, pages 93–104, 2006.

[150] Dennis Pagano and Walid Maalej. How do open source communities blog? *Empirical Software Engineering*, 18(6):1090–1124, 2013.

[151] Fabio Palomba. Textual analysis for code smell detection. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 769–771. IEEE, 2015.

[152] 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*, pages 281–290. IEEE, 2015.

[153] Chris Parnin and Christoph Treude. Measuring API documentation on the web. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, pages 25–30, New York, NY, USA, 2011. ACM.

[154] Chris Parnin, Christoph Treude, and Margaret-Anne Storey. Blogging developer knowledge: Motivations, challenges, and future directions. In

Proceedings of the 21st International Conference on Program Comprehension, pages 211–214, Piscataway, NJ, USA, 2013. IEEE Press.

[155] Luca Pascarella and Alberto Bacchelli. Classifying code comments in java open-source software systems. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 227–237, Piscataway, NJ, USA, 2017. IEEE Press.

[156] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. Vulnerable open source dependencies: counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 42. ACM, 2018.

[157] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[158] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 426–437, 2015.

[159] Roxana Lisette Quintanilla Portugal and Julio Cesar Sampaio do Prado Leite. Extracting requirements patterns from software repositories. In *Proceedings of the 24th International Requirements Engineering Conference Workshops*, pages 304–307, Piscataway, NJ, USA, 2016. IEEE.

[160] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the content of github readme files. *Empirical Software Engineering*, 24(3):1296–1327, 2019.

[161] Philips Kokoh Prasetyo, David Lo, Palakorn Achananuparp, Yuan Tian, and Ee-Peng Lim. Automatic classification of software related microblogs. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 596–599. IEEE, 2012.

[162] Parvati Raghuram, Clem Herman, Esther Ruiz-Ben, and Gunjan Sondhi. Women and it scorecard-india. https://www.nasscom.in/knowledge-center/publications/women-and-it-scorecard-2017.

[163] Akond Rahman, Effat Farhana, and Nasif Imtiaz. Snakes in paradise?: insecure python-related coding practices in stack overflow. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 200–204. IEEE Press, 2019.

[164] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 432–441. IEEE Press, 2013.

[165] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 61:1–61:11. ACM, 2012.

[166] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 349–359. IEEE, 2016.

[167] Yolanda A. Rankin and Jakita O. Thomas. Straighten up and fly right: Rethinking intersectionality in hci research. *Interactions*, 26(6):64–68, October 2019.

[168] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[169] 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*, pages 155–165, 2014.

[170] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[171] Penny Rheingans, Erica D'Eramo, Crystal Diaz-Espinoza, and Danyelle Ireland. A model for increasing gender diversity in technology. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 459–464, 2018.

[172] 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. *Management science*, 52(7):984–999, 2006.

[173] Gregorio Robles, Laura Arjona Reina, Jesús M González-Barahona, and Santiago Dueñas Domínguez. Women in free/libre/open source software: The situation in the 2010s. In *IFIP International Conference on Open Source Systems*, pages 163–173. Springer, 2016.

[174] Daniele Romano and Martin Pinzger. Using source code metrics to predict change-prone java interfaces. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 303–312. IEEE, 2011.

[175] Pamela H Russell, Rachel L Johnson, Shreyas Ananthan, Benjamin Harnke, and Nichole E Carlson. A large-scale analysis of bioinformatics code on github. *PLoS One*, 13(10):e0205898, 2018.

[176] Johnny Saldaña. *The Coding Manual for Qualitative Researchers*. SAGE Publications, 2009.

[177] Lucía Santamaría and Helena Mihaljević. Comparison and benchmark of name-to-gender inference services. *PeerJ Computer Science*, 4:e156, 2018.

[178] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[179] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 771–781. IEEE, 2012.

[180] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. Cataloging GitHub repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319, New York, NY, USA, 2017. ACM.

[181] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE transactions on software engineering*, 37(6):772–787, 2010.

[182] Yonghee Shin and Laurie Williams. An initial study on the use of execution complexity metrics as indicators of software vulnerabilities. In *Proceedings of the 7th International Workshop on Software Engineering for Secure Systems*, pages 1–7, 2011.

[183] Yonghee Shin and Laurie Williams. Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering*, 18(1):25–59, 2013.

[184] NC Shrikanth and Tim Menzies. Assessing practitioner beliefs about software defect prediction. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 182–190. IEEE, 2020.

[185] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the International Symposium on the Foundations of Software Engineering*, pages 23–34, New York, NY, USA, 2006. ACM.

[186] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.

[187] Vandana Singh. Women participation in open source software communities. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, page 94–99, New York, NY, USA, 2019. Association for Computing Machinery.

[188] Vandana Singh and William Brandon. Open source software community inclusion initiatives to support women participation. In Francis Bordeleau, Alberto Sillitti, Paulo Meirelles, and Valentina Lenarduzzi, editors, *Open Source Systems*, pages 68–79, Cham, 2019. Springer International Publishing.

[189] Qinbao Song, Yuchen Guo, and Martin Shepperd. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12):1253–1269, 2018.

[190] Andrea Di Sorbo, Sebastiano Panichella, Corrado A. Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. Development emails content analyzer: Intention mining in developer discussions (t). In *Proceedings of the 30th International Conference on Automated Software Engineering*, pages 12–23, Piscataway, NJ, USA, 2015. IEEE Press.

[191] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 908–911. ACM, 2018.

[192] Charles Spearman. The proof and measurement of association between two things. *American journal of Psychology*, 15(1):72–101, 1904.

[193] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1379–1392, 2015.

[194] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering*, pages 273–284, 2016.

[195] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barri-

ers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering*, pages 273–284, New York, NY, USA, 2016. ACM.

[196] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, 2017.

[197] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: pull request acceptance of women versus men. *PeerJ Computer Science*, 3:e111, May 2017.

[198] Emma G Thomas, Bamini Jayabalasingham, Tom Collins, Jeroen Geertzen, Chinh Bui, and Francesca Dominici. Gender disparities in invited commentary authorship in 2459 medical journals. *JAMA network open*, 2(10):e1913682–e1913682, 2019.

[199] Ferdian Thung. Api recommendation system for software development. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 896–899. IEEE, 2016.

[200] Ferdian Thung, Stefanus A Haryono, Lucas Serrano, Gilles Muller, Julia Lawall, David Lo, and Lingxiao Jiang. Automated deprecated-api usage update for android apps: How far are we? In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 602–611. IEEE, 2020.

[201] Rebecca Tiarks and Walid Maalej. How does a typical tutorial for mobile development look like? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 272–281, New York, NY, USA, 2014. ACM.

[202] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web? (NIER track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 804–807, New York, NY, USA, 2011. ACM.

[203] Christoph Treude, Fernando Figueira Filho, and Uirá Kulesza. Summarizing and measuring development activity. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 625–636, New York, NY, USA, 2015. ACM.

[204] Christoph Treude and Martin P. Robillard. Augmenting API documentation with insights from Stack Overflow. In *Proceedings of the 38th International Conference on Software Engineering*, pages 392–403, New York, NY, USA, 2016. ACM.

[205] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Anita Sarma, and Igor Steinmacher. Hidden figures: Roles and pathways of successful oss contributors. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW2), October 2020.

[206] Asher Trockman. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, page 524–526. ACM, May 2018.

[207] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering*, pages 511–522. ACM, 2018.

[208] Michele Tufano, Fabio Palomba, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia, and Denys Poshyvanyk. When and

why your code starts to smell bad. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 403–414. IEEE, 2015.

[209] Marie E Vachovsky, Grace Wu, Sorathan Chaturapruek, Olga Russakovsky, Richard Sommer, and Li Fei-Fei. Toward more gender diversity in cs through an artificial intelligence summer program for high school girls. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 303–308, 2016.

[210] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Perceptions of diversity on github: A user survey. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 50–56. IEEE, 2015.

[211] 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*, pages 3789–3798, 2015.

[212] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. A data set for social diversity studies of github teams. In *2015 IEEE/ACM 12th working conference on mining software repositories*, pages 514–517. IEEE, 2015.

[213] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. License usage and changes: a large-scale study on github. *Empirical Software Engineering*, 22(3):1537–1577, 2017.

[214] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, War-

ren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020.

[215] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W Wallin. Carrots and rainbows: Motivation and social practice in open source software development. *MIS quarterly*, pages 649–676, 2012.

[216] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. Deep semantic feature learning for software defect prediction. *IEEE Transactions on Software Engineering*, 46(12):1267–1293, 2018.

[217] Elaine J. Weyuker, Thomas J. Ostrand, and Robert M. Bell. Using developer information as a factor for fault prediction. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, page 8–8. IEEE, May 2007.

[218] Brian Witten, Carl Landwehr, and Michael Caloyannides. Does open source improve system security? *IEEE Software*, 18(5):57–61, 2001.

[219] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[220] Xin Xia, Yang Feng, David Lo, Zhenyu Chen, and Xinyu Wang. Towards more accurate multi-label software behavior learning. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014*

Software Evolution Week-IEEE Conference on, pages 134–143. IEEE, 2014.

[221] Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. An empirical study of security issues posted in open source projects. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[222] Tao Zhang, Jiachi Chen, He Jiang, Xiapu Luo, and Xin Xia. Bug report enrichment with application of automated fixer recommendation. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 230–240. IEEE, 2017.

[223] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. Detecting similar repositories on GitHub. In *Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering*, pages 13–23, Piscataway, NJ, USA, 2017. IEEE.

[224] Yun Zhang, David Lo, Xin Xia, Bowen Xu, Jianling Sun, and Shanping Li. Combining software metrics and text features for vulnerable file prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 40–49. IEEE, 2015.

[225] Shurui Zhou, Stefan Stanciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wasowski, and Christian Kästner. Identifying features in forks. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 105–116. IEEE, 2018.

[226] Yaqin Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 914–919. ACM, 2017.

[227] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 995–1010, 2019.

[228] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 421–428. IEEE, 2010.

[229] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, 2010.

[230] Frances Zlotnick. Github open source survey 2017. `http://opensourcesurvey.org/2017/`, June 2017.

[231] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering*, 46(8):836–862, 2018.

Appendix A

Literature Tables

Table A.1: Comparison of research questions, goals, and findings of closely related literature on gender and geographic diversity.

	Research questions or goals	**Findings**
Geographic, Gender Inclusion (Our Work)	Identify how gender diversity has changed over time and across regions. Identify gender-based and geographic-based developer participation	Noticeable differences across gender diversity in regions specifically, Asia and Americas being the highest Barriers and motivations to contributing converge across geographic regions No strong correlation between gender and geographic diversity
The Shifting Sands of Motivation [67]	Identify what motivates OSS contributors and how contributors' motivation changes as OSS matured (e.g., contributors themselves gaining tenure)	Main motivations differ between novice contributors and experienced ones Motivations such as learning and knowledge sharing remained important overtime. While altruism increased in importance and self-serving usage decreased in importance.
Women Core Developers [25]	Identify the gender diversity and work practices of core developers in OSS communities	OSS has horizontal and vertical segregation No significant differences between work practices of men v. women
Diversity, Where Do We Stand [18]	Determine the level of gender diversity among popular OSS projects	Women make up less than 10% of core contributors No significant difference among men vs. women on selected projects
Diversity Teams Study [147]	Understand the impact of gender and nationality diversity on team productivity and collaboration quality	Higher gender diversity→ lower team average issue fixing time Nationality diversity → lower team politeness

Table A.2: Comparison of research methods, population / initial sample, and participants' data of closely related literature on gender and geographic diversity.

	Research methods	Population / initial sample	Participants / data
Geographic, Gender Inclusion (Our Work)	Mixed methods study of GitHub projects over time and conducting a purposefully sampled survey with developers across geographic regions and identifiable genders (men, women and unidentifiable)	125,485,095 projects from GHTorrent	21,456 repositories, 70,621 commit authors, 122 survey respondents across 5 large geographic regions and across genders
The Shifting Sands of Motivation [67]	Survey of OSS contributors recruited from social media sites (e.g., Twitter, Facebook, Reddit, LinkedIn, and Hackernews, through groups related to OSS development, and personal contacts	Open to all self-reported OSS contributors. Filtered on reported experience and response validity	242 responses from 5 different continents. Includes 82% men, 81% coders, 26% report being paid for contributions
Women Core Developers [25]	Mixed methods study of mining software repositories, identifying gender of contributors and interviewing women core developers	Top 100 most popular projects written in the top 15 most popular programming languages	711 projects, 35 women core developers
Diversity, Where Do We Stand [18]	Mined code review repositories of the top 10 popular OSS project on GitHub	Top 10 OSS Projects using Gerrit and had at least 15,000 code reviews	683,865 pull requests, 4543 non-casual contributors
Diversity Teams Study [147]	Built regression models comparing collaboration in issues and dialogue of politeness	2014 GHTorrent dataset scoped to closed issues with 2 comments	33,673 issues with 71,423 comments posted by 13,872 developers

Appendix B

Survey Results: Encouraging Women

We asked all survey respondents(regardless of identified gender) about what they think can encourage more women in GitHub and received 73 responses to this questions. We qualitatively analyzed responses to this question via open coding and axial coding process which included iterative review of our themes. We grouped response according to regions, but despite using our strategic sampling approach, we did not have a critical mass of responses across regions and genders to make meaningful conclusions. We provide summaries of responses by the following themes below supplemented with quotes.

B.1 Encouragement through awareness

We received several broader responses about how to encourage women through providing awareness via several activities. *"Awareness will go a long way in encouraging women to participate. A lot of people would love to participate but wasn't sure where to start."* (P14)

One activity mentioned was making education more accessible: *"Accessible education. I think many women and girls don't realise they have the*

skills needed to contribute to programming & software projects. Teaching young women that they have the potential to do this is really important." (P13)

We also see several quotes about fostering a more welcoming community that includes fewer misogynistic developers and more overall encouragement: *"Fewer misogynistic developers."* (P25) *"De-stigmatize programming as a male dominated profession"* (P86)

Although strategies in this category were rather general, they indicated that respondents are familiar with challenges some women face in OSS communities.

B.2 Creating opportunities

Respondents also indicated concrete recommendations on what may support encouraging women on GitHub. Some of these recommendations include adding events that specifically support women such as having women-centered events or even amplifying the presence of women that are active in the community: *"A women support circle is nice, I've seen effort and took part in some but I find women are more comfortable and more encouraged within the same gender group. "* (P17) *"Show what women who are working with this are doing and how is their experience, do projects/workshops."* (P35) *"More women reviewers. More women acting directly on the governance of large open source projects."* (P52)

Likewise other participants mentioned that the community should emphasize use of existing mechanisms such as project codes of conduct: *"Enforce codes of conduct"* (P79) *"more welcoming in projects, a well-defined code of conduct to make them feel more comfortable"* (P101) *"Giving more visibility and fighting against bad behaviors by other men"* (P87)

We also had participants cite that encouraging women to be apart of a transparent developer sprint and workshops which will provide more clarity to the multiple phases of the contribution process: *"Encourage more open de-*

vsprints and workshops to help women get started easily. More hands-on sessions on upstream contributions." (P37) *"Creating awareness among women contributors and building confidence, by conducting interactive sessions on open source and contributions"* (P47)

Other broader recommendations participants made were to fund developers making contributions, *"Getting paid"* (P19), and being more transparent in the code review process, *"being more articulate about feelings and motivation behind some critique that could come up for example in code review"* (P51).

B.3 Outside of GitHub

Many respondents reported that this solution is more of a broader issue that is broader than GitHub. Some participants reported that there should be a focus beyond computer science and on STEM fields in general: *"I guess you need help more women go to colleges and learn STEM, and make sure they will not be rejected from some professional jobs in the STEM field after graduation."* (P96) *"We just need more women in programming overall, and I think school outreach programs are the best thing."* (P64)

Another set of respondents indicated that changing the bro-culture of technology in the software world: *"Don't see why Github has anything to do with women's participation. In general, low participation to oss from women's may be related to they being minority in the whole bro cultured software world."* (P88) *"Men should know how to interact with women without an air of authority and welcoming. .."* (P31)

Some respondents described that there are broader global issues that persist outside of work: *"Global women's rights, not in IT only."* (P34) *"Treating women as equals"* (P33)

These findings indicate that respondents were aware of challenges in tech but also it made sense to address issues at a wider scale.

B.4 The "I Don't Care"s

Finally, we did receive several responses that either dismissed this focus on experience of women in OSS, were unclear on the challenges that women face or actively responded with a negative tone to this question (as opposed to simply leaving this optional question blank). As we have not seen in previous literature where this negative sentiment towards empowering a marginalized group has been acknowledge, we found it imperative to share the responses we received here: *"It is naturally that women are less interesting in technologies than men. I don't see any barrier to prevent women to participating in Github."* (P109) *"I don't know why this is even a thing."* (P41) *"Ask the women. I'm not stopping them."* (P9)

We also had several toxic recommendations suggesting women look into *"Sex reassignment surgery"* (P53) and that *"Good engineers should help themselves"* (P24). We highlight these responses not to amplify these biased perspectives, but to show that there are OSS contributors in the community who *do not* understand that there is an issue with the gender diversity. It is not the job of the marginalized contributors to 'fix' the community—it is up to *everyone* to create an inclusive environment. Future work should explore interventions that create a broader awareness of why it is important for everyone to be inclusive along gender and regional diversity.

We hope that these responses encourage researchers to study a variety of gender experiences (including non-binary genders) to capture rich-region specific diversity issues. Having more region-specific studies will allow us to provide bespoke solutions that take into the cultural nuance of each region.