

Ownership and Accountability in Software Teams

UMME AYMAN KOANA

A THESIS SUBMITTED TO THE FACULTY OF
GRADUATE STUDIES IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL AND
COMPUTER ENGINEERING

YORK UNIVERSITY
TORONTO, ONTARIO

JUNE, 2023

© **UMME AYMAN KOANA, 2023**

Abstract

Ownership of software artifacts has become a point of interest to software teams. Researchers modeled ownership of software artifacts with different models and in relationship with a variety of code and developers' performance metrics. These models have been evaluated for both propriety and open-source software. Bightsquid is a software startup that provides a healthcare communication system. At the time of the COVID-19 pandemic (starting March 2020), the company actively modified its processes to improve developers' experience and accountability. As the provider of health communications, Brighsquid was receiving an amplified number of user requests to accommodate the changing needs in the health care system. Yet, the management team observed the lack of accountability among the developers in accepting and finalizing these requests. The company changes the task assignment process to the team with the main motivation to increase developers' accountability.

Motivated by this problem statement and the status of the partnered company, this thesis presents four main contributions: a systematic literature review on software ownership, a case study with Brighsquid to compare their ownership status with existing research, an evaluation of the impact of enhanced accountability through a comparative analysis of issue assignment models, and a survey of software developers to explore the broader relationship between accountability and ownership.

Acknowledgments

I would like to express my sincere gratitude to Dr. Maleknaz Nayebi for her invaluable guidance as my supervisor throughout my MASc thesis. Her expertise and support have greatly influenced the quality and direction of my research. I am also thankful to Shadikur Rahman, my husband, for his unwavering support and encouragement. I would like to acknowledge the significant contributions of Quang Hy Le, an undergraduate student, whose assistance and collaboration greatly enhanced my research. Lastly, I extend my thanks to my family and friends for their constant support. Without them, this achievement would not have been possible.

Preface

The research presented in this thesis has been conducted in collaboration between Brightsquid and EXINES Lab led by Dr. Maleknaz nayebi. This thesis is organized in paper format following the guidelines for paper-based theses. Parts of this report are undergoing review by the following venues:

- Koana, U., Le, Q., Rahman, S., Carlson, C., Chew, F., & Nayebi, M. (2023). Examining Code Ownership in Software Teams: A Systematic Literature Review and Replication Study at Brightsquid. In 2023 Empirical Software Engineering (ESE). (submitted)
- Koana, U., Chew, F., Carlson, C., & Nayebi, M. (2023). Ownership in the Hands of Accountability: A case study and a Developer survey. In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2023 - Industry track. (submitted)

Table of Contents

| | |
|---|------------|
| Abstract | ii |
| Acknowledgments | iii |
| Preface | iv |
| Table of Contents | v |
| List of Tables | ix |
| List of Figures | ix |
| Chapter 1: Introduction | 1 |
| 1.1 Ownership in Software Teams | 2 |
| 1.2 Developers' Accountability and Productivity | 2 |
| 1.3 Task Assignment in Software Teams | 3 |
| 1.4 Motivation and Objectives | 4 |
| 1.5 Thesis Outline | 6 |
| Chapter 2: Examining Code Ownership in Software Teams: A Sys- | |
| tematic Literature Review and Replication Study at Brightsquad | 8 |
| 2.1 Introduction | 9 |
| 2.2 Protocols Used for Systematic Literature Review | 13 |
| 2.3 Protocols Used for Benchmarking and Replication | 17 |
| 2.4 Result of Systematic Literature Review (RQ1, RQ2, RQ3) | 19 |

| | | |
|-------|---|----|
| 2.4.1 | Ownership Definitions and Models (RQ1) | 20 |
| 2.4.2 | Ownership Modeling Treatments and Outcomes (RQ2) | 29 |
| 2.4.3 | Analytical Model, Replicability, and Evaluation setup (RQ3) | 32 |
| 2.5 | Benchmarking State of the Art Model in Brighsquid | 37 |
| 2.5.1 | Hypothesis and Research Methodology in Original Studies | 39 |
| 2.5.2 | Empirical Data from Brightsquid | 42 |
| 2.5.3 | Empirical Results and Ownership Status Quo at Brightsquid | 44 |
| 2.5.4 | Comparing results between Brightsquid and the reported state of practice | 53 |
| 2.6 | Discussion and Implication of Findings at Brightsquid | 57 |
| 2.7 | Threats to Validity | 62 |
| 2.8 | Conclusion | 64 |

Chapter 3: Ownership in the Hands of Accountability at Brightsquid:

| | | |
|-------|---|-----------|
| | A case study and a Developer survey | 66 |
| 3.1 | Introduction | 67 |
| 3.1.1 | Developer Accountability in the Face of High Customer Demand | 68 |
| 3.1.2 | Research Questions | 70 |
| 3.2 | Brighsquid Data | 71 |
| 3.3 | Background | 73 |
| 3.4 | Empirical Methodology | 75 |
| 3.4.1 | The relationship between code ownership and code quality met- rics (RQ1) | 75 |
| 3.4.2 | Bug predictability performance (RQ2) | 76 |
| 3.4.3 | Developer’s perception of ownership and accountability in soft- ware teams (RQ3) | 77 |
| 3.5 | Results | 78 |

| | | |
|-------------------|---|------------|
| 3.5.1 | Relationship between code ownership and code quality metrics (RQ1) | 78 |
| 3.5.2 | Bug predictability performance (RQ2) | 82 |
| 3.5.3 | Survey with developers (RQ3) | 85 |
| 3.6 | Discussion | 89 |
| 3.6.1 | Assignment process and relation to accountability, ownership, and code quality | 90 |
| 3.6.2 | Survey Implications (RQ3) | 92 |
| 3.7 | Threats to validity | 93 |
| 3.8 | Related Work | 94 |
| 3.8.1 | Individual and team accountability | 94 |
| 3.8.2 | Ownership in Software Teams | 95 |
| 3.9 | Conclusion | 96 |
| Chapter 4: | Conclusion and Future Work | 97 |
| 4.1 | Conclusion | 97 |
| 4.2 | Future Work | 98 |
| | Bibliography | 101 |

List of Tables

| | | |
|------|---|----|
| 1.1 | Research Questions addressed in our thesis | 6 |
| 2.1 | Search string and number of returned records within each database . | 13 |
| 2.2 | Inclusion and Exclusion criteria for our study | 15 |
| 2.3 | Types and definition of corporal ownership in software engineering state of the art (RQ1). | 23 |
| 2.4 | Independent, Dependent, and controlled variables used “more than once” in modeling ownership studies. | 30 |
| 2.5 | Data availability within our systematic literature review. ✕shows the unavailability of the data while ✓represents data availability. | 34 |
| 2.6 | Research Questions addressed in the three replicated studies | 38 |
| 2.7 | What, who, and How ownership was discussed in the selected studies for replication and the dependent, and control variables. | 40 |
| 2.8 | Independent variable in the selected studies for replication. | 41 |
| 2.9 | Characteristics of the Brightsquid projects active as of March 2020 and included in our dataset. | 43 |
| 2.10 | Spearman correlation between metrics and the number of bugs on file level granularity. Bold and underlined values are absolute correlation coefficients above 0.50 and 0.75, respectively | 46 |
| 2.11 | Spearman correlation between metrics and bug numbers on directory level. Bold values are absolute correlation coefficients above 0.50. . . | 47 |

| | | |
|------|--|----|
| 2.12 | Variance in failures for the base model (Code metrics) which includes standard metrics of COMPLEXITY, SIZE, and CHURN, as well as the models with MINOR, MAJOR and OWNERSHIP added | 49 |
| 2.13 | Details on Precision, Recall, and F-measure for predicting defective source files and directories. | 51 |
| 2.14 | Metric importance for prediction model based on ownership metrics classifying detective source files | 53 |
| 2.15 | Metric importance for prediction model based on ownership metrics classifying source directories | 53 |
| 3.1 | Characteristics of the studied Brightsquad projects. | 73 |
| 3.2 | Definition of metrics inferred from literature and used in our study. . | 74 |
| 3.3 | Spearman correlation between metrics and bug numbers on file level. | 78 |
| 3.4 | Spearman correlation coefficients between metrics and the number of bug fixes on directory level. | 80 |
| 3.5 | Variance in bugs for the Base model (Code metrics) and models with Minor, Major and Ownership added. An asterisk* denotes that a model showed models statistically significant improvement when the additional variable was added. | 81 |
| 3.6 | Details on Precision, Recall, and F-measure for predicting defective files. | 82 |
| 3.7 | Metric importance in accordance to ownership metrics across the changed files | 83 |
| 3.8 | Metric importance in accordance to ownership metrics across the changed directories | 84 |
| 3.9 | Source and degree of developer's sense of ownership | 87 |

| | | |
|-----|--|-----|
| 4.1 | Specifications of the papers directly relevant to "Ownership" of software artifacts included in our systematic literature review. Paper which did not discuss ownership directly (such as S66) are excluded from this table. | 115 |
|-----|--|-----|

List of Figures

| | | |
|------|--|----|
| 2.1 | Study Selection process for our systematic literature review | 13 |
| 2.2 | Process of replicating studies for Brightsquid | 19 |
| 2.3 | Total number of publications per year. Most papers have been published in recent years which makes this study very timely. | 20 |
| 2.4 | Taxonomy of the nature (What?), the owner (who?), and the degree of ownership (How?) as the result of our systematic review. | 21 |
| 2.5 | Treatment-outcome relation as we studied in this SLR. The process and the diagram are adopted from the work of Wohlin et al. [45]. . . | 29 |
| 2.6 | Distribution of paper based on (a) Type of analytics and (b) Research Method | 33 |
| 2.7 | Percentage of papers which their code and data are available | 33 |
| 2.8 | Descriptive statistics of ownership metrics of Brightsquid projects. The metrics calculated based on Greiler et al. [68] work. | 44 |
| 2.9 | Main steps taken in the original studies of Bird et al. [6][S7], Greiler et al. [68][S16], and Foucault et al. [74] [S19] to analyze ownership . . | 45 |
| 2.10 | Relative importance of metrics in regression models using the NUMBER OF BUGS as the dependent variable. | 50 |
| 3.1 | The process of (a) product manager details tasks for every user scenario and assign to a developer - Before March 2020 (b) product manager assign user story to a developer who is then responsible to design solution and define tasks. | 69 |

| | | |
|-----|--|----|
| 3.2 | JIRA issue hierarchy of Brightsquid | 72 |
| 3.3 | Demographics of survey participants | 85 |
| 3.4 | Accountability Levels of Developers in a Team. Participants' account- ability were measured using a Likert scale. | 86 |
| 3.5 | Who should be held accountable for maintaining an artifact with shared ownership? | 88 |
| 3.6 | Developers' perceived degree of relationship between accountability and ownership (5 = strongest degree). | 89 |

Chapter 1

Introduction

The concept of software ownership has long been a topic of discussion in the software engineering literature, particularly in relation to code quality. While the literature offers a wealth of knowledge on the subject, there is a lack of standardized benchmarks for comparison. Recognizing this gap, we partnered with an industry organization to conduct an evaluation that aims to compare code ownership practices with the state-of-the-art approaches, as well as examine how different task assignment models impact code ownership and quality. Our collaboration with Brightsquad, a company specializing in secure communication solutions for the healthcare industry, arose from their growing concern regarding developer ownership and accountability [1]–[5]. In a collaborative project, Brightsquad was interested to analyze their current situation in order to strike a balance between fostering collaboration among developers while ensuring individual accountability. To accomplish this, our study entails an analysis of ownership models in software development and investigates the relationship between ownership and accountability. We then conducted a survey with 67 developers to evaluate the extent these findings are applicable outside this particular case study with Brightsquad.

This thesis is based on two manuscripts, one includes a holistic systemic literature review on ownership and dependent, control and independent variables. Hence, we only briefly explain the main concepts of ownership and accountability in this section

and further discuss the objectives of this thesis and moving forward to presenting each paper.

1.1 Ownership in Software Teams

In the field of Software Engineering, ownership refers to the possession, control, or rights associated with artifacts, specifically software code. The concept of ownership has been widely explored in relation to different performance metrics of projects, products, and teams. Ownership within software teams is of great importance in cultivating dedication, initiative, and ensuring the delivery of high-quality work [6],[7]. Within the software industry, two primary forms of ownership have been the subject of discussion: Psychological Ownership, which relates to an individual's personal sense of possession [8], and Corporal Ownership, which is associated with the developmental history of software artifacts such as code, tasks, bugs, and products etc. Corporal Ownership, in particular, holds significant popularity within the realm of software development.

1.2 Developers' Accountability and Productivity

Accountability is an obligation or willingness to accept responsibility or to account for one's actions¹. In the software development industry, it refers to the responsibility of individuals or teams for their actions, decisions, and outcomes related to the development process [9]. Ownership has a significant impact on both the organization and the individual and implementing effective accountability systems can lead to positive outcomes in terms of task performance [10]. When accountability is shared among team members, it brings about added risks for individuals as if a team member fails to fulfill their responsibilities, it creates challenges for the entire team to accomplish their tasks effectively [11].

¹<https://www.merriam-webster.com/dictionary/accountability>

Another important concept in software development is the developer’s productivity which is about how efficiently and effectively developers get their work done in software development teams. In the context of software development, organizations measure productivity by looking at the amount of work and time invested in producing software [12]. Productivity in software development extends beyond individual performance. If a developer focuses solely on their personal productivity, it can negatively impact the overall productivity of the team [13]. Traditionally, managers measure the productivity of software developers through the number of lines of code or function points implemented per hour worked, aiming to reduce costs, enhance quality, and accelerate software development[14] but developers consider their days productive when they can successfully finish numerous or big tasks without facing significant interruptions or context switches [15]. Recently, the COVID-19 pandemic has led to a widespread discussion on productivity as companies implemented Work From Home (WFH) arrangements. This shift has had a significant impact both positive and negative on developer productivity [16].

1.3 Task Assignment in Software Teams

Team culture and project management practices have an impact on task assignments. In both open source software (OSS) development and agile software development, team culture holds significant importance as these approaches heavily rely on collaboration and cooperation among developers [17]. Agile practices often refer to the collective and voluntary modes of task assignment where every developer is the owner of the team’s “product” [18]. This task ownership holds significant importance as it makes team members more responsible for their own tasks which plays an important factor in the progress and success of the project [19]. Usually, the team leader takes on the responsibility of assigning tasks and bugs, dedicating considerable time to determining the most appropriate developer for each assignment [20]. However, if a task is assigned to an unsuitable team member, it not only compromises the quality

of the outcome but also results in the wastage of valuable resources like time, money, and the trust of the client [21]. For these, Task allocation involves considering various factors such as technical expertise, time zone differences, resource cost, task dependencies, vendor reliability, and task size. Identifying these factors is crucial to ensure that important aspects are not overlooked when devising task assignment strategies [22]. Task prioritization is another critical responsibility for team leaders as it ensures that the company’s key goals remain in focus and can be successfully achieved. It serves as the foundation for a successful project, enabling effective allocation of resources and alignment with strategic objectives [23]. Several studies have addressed the challenges related to task assignment in software development teams and have proposed various approaches to tackle them [24], [25], [26].

1.4 Motivation and Objectives

Our partner company Brightsquid is a global provider of HIPAA-compliant communication solutions providing messaging, email, and large file transfer for medical and dental professionals since 2009. Their comprehensive range of services includes messaging, email, and secure large file transfer, empowering healthcare communities to aggregate, generate, and share protected health information seamlessly. When we performed our study Brightsquid had a total of 39 projects, with a diverse team of 52 developers who have contributed to the company from 2009 to 2022.

One notable characteristic of the Brightsquid team is their distributed nature, allowing developers to work remotely from different locations, leveraging the company’s online-based operations. As part of our collaboration, Brightsquid wanted to evaluate their code ownership status, recognizing the vital connection between ownership and software quality. They also expressed interest in benchmarking their status against other companies that have previously discussed and analyzed ownership dynamics.

Furthermore, as a healthcare service provider during COVID-19 pandemic the user requests increased significantly. To cope up with excessive market demands and user

needs Brightsquid realized that they need to expand their developer’s accountability toward end-customers. Despite maintaining their existing distribution approach, they proactively responded to the overwhelming customer demands and evolving market needs by introducing a new issue assignment model to enhance developer accountability where developers were fully responsible to execute and deliver a user request to end users instead of the product manager. Motivated by the change of issue assignment model, Brightsquid aimed to assess the impact of their revised approach, specifically focusing on the relationship between expanded accountability and ownership within the company. Additionally, they expressed a broader interest in understanding developers’ perspectives on the relationship between ownership and accountability outside of Brightsquid. This prompted us to:

- Perform a systematic literature review on the software engineering ownership model, evaluations, and their applicability status.
- Evaluate and compare the overall status of ownership of Brightsquid with the state-of-art research.
- Evaluate the status of ownership with the current and previous issue assignment model of Brightsquid.
- Qualitatively evaluate the relation between ownership and accountability through a developer survey.

The research questions addressed in our thesis are presented in Table 1.1 where RQ1, RQ2 and RQ3 were answered through a systematic literature review. We replicated three studies identified in the literature review using Brightsquid data to address RQ4. Then, for RQ4 and RQ5, we collected Brightsquid data from March 2018 to March 2020 and March 2020 to March 2022 to analyze the impact of new issue assignment model with enhanced accountability. Finally, we conducted a developer study

Table 1.1: Research Questions addressed in our thesis

| Research Questions | Type | Chapter |
|---|----------------------------|---------|
| RQ1: How is ownership defined and modeled in software engineering literature? | Literature Review | 2 |
| RQ2: What treatment and outcomes have been used in modeling ownership in software engineering literature? | Literature Review | 2 |
| RQ3: What analytics and research methods have been employed in the study of ownership in software engineering, and to what extent are these studies replicable? | Literature Review | 2 |
| RQ4: How does the code ownership status in Brightsquid compare with the findings of state-of-the-art studies? | Replication and case study | 2 |
| RQ5: How has the relationship between ownership and software quality metrics changed following the implementation of the new issue assignment model at Brightsquid? | Case Study | 3 |
| RQ6: How did the revision of BrightSquid’s issue assignment process affect the performance of prediction models in identifying defective files and directories? | Case Study | 3 |
| RQ7: How do developers perceive the relationship between accountability and ownership? | Survey | 3 |

to explore different aspects of ownership and accountability and their relationship for addressing RQ4.

1.5 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 presents the systematic literature review on the software ownership model (RQ1 -RQ3) and reports the results of the ownership replication analysis of Brightsquid (RQ4). To address RQ1, we collected and combined definitions from various research papers related to ownership in software engineering. This includes identifying the type of artifact being discussed, the role of the artifact owner in its maintenance, and the degree of ownership indicating the rights, responsibilities, and involvement of team members in maintaining the artifact. To answer RQ2, We made a comprehensive list of artifacts and outcomes associated with ownership models and found that bug is the most frequently considered outcome in the software industry. Then in RQ3, we analyzed the ownership

model type (descriptive, diagnostic, predictive, or prescriptive), data availability, and replicability of our selected studies. We replicated the three most relevant studies from our literature review to evaluate the status of code ownership in Brighsquad. We applied ownership models of those studies to nine active projects of Brighsquad for addressing RQ4.

In Chapter 3, We conducted a code change analysis of Brightsquad to investigate the effects of previous and current issue assignment models (RQ5, RQ6). For RQ5, we examined the relationship between code ownership metrics and code quality, specifically how this relationship changed with the adoption of the new model. To address RQ6, we compared the performance of a classification model in predicting defects before and after the process change, as well as evaluated the importance of metrics in predicting defects. Additionally, we surveyed 67 developers to gain insights into their perceptions of accountability and ownership (RQ7).

Finally, we conclude our study at Chapter 4 summarizing our key findings, contributions and future work.

Chapter 2

Examining Code Ownership in Software Teams: A Systematic Literature Review and Replication Study at Brightsquid

The effective ownership of software artifacts, especially code, is vital for ensuring accountability, sharing knowledge, and enhancing code quality. Researchers have proposed models to establish ownership of software artifacts and their relationship with developer performance and code quality. However, these models have mostly been evaluated on open-source repositories. Further, there is a lack of systematic studies on code ownership in proprietary software. Our study aims to systematically study the different ownership models and assess the current ownership status of our partner company, Brightsquid.

During our collaboration, the software engineering team at Brightsquid expressed interest about comparing their code quality against ownership factors in other teams. We conducted a systematic literature review to establish a comparative baseline and identified 79 relevant papers published between 2005 and 2022. We created a taxonomy of ownership artifacts based on ownership artifacts, type of owners, and degree of ownership. Also, we compiled a list of modeling variables in software ownership and examined the type of analytics used in each study. We also evaluated the replication status of each study. As a result, we identified nine distinct software artifacts whose

ownership has been discussed in the literature, with "Code" as the most frequently analyzed artifact. We found that only 3 papers (3.79%) provided both code and data, while 9 papers (11.4%) provided only data. After identifying the most relevant studies in partnership with Brightsquid, we conducted a replication study using their data. By comparing our findings with those reported in state-of-the-art research, we observed similar results in terms of the correlation between stronger file ownership and a lower number of bugs at the file level, as well as the correlation between the number of major owners and a higher number of bugs. However, unlike previous studies, our observations did not show a strong predictive power for code ownership metrics when it comes to predicting code quality.

This study provides a holistic overview of the ownership models. Our study provided taxonomies and the list of dependent, independent, and controlled variables used for modeling ownership provides guidance for future research on this topic. Finally, the findings of this study can also help inform the development of best practices for software development teams and guide decisions related to code ownership and collaboration.

2.1 Introduction

Ownership is the act, state, or right of possessing something. Within the Software Engineering field, the ownership of artifacts and, in particular, software code has been discussed in various ways and in relation to a variety of performance metrics of the project, product, and team. Yet, there is no systematic review of the existing models and the status and availability of the replication packages, benchmarks, and best practices. Further, there is no benchmark on software ownership acting as a gold standard. In a collaborative project with a software company, Brightsquid¹, we were interested in evaluating the ownership status within the company and performing a comparison with the status in other companies.

¹<https://Brightsquid.com/>

Brighsquid is a provider of healthcare practice data security services. **Brighsquid** solutions are HIPPA² compliant, enabling secure messaging and large file transfer for dental and medical professionals. Their communication platform supports aggregating, generating, and sharing protected health information across communities of healthcare patients, practitioners, and organizations. **Brighsquid** has been working on a number of projects to achieve these business goals. The company has overall 39 separate software repositories at different stages of the life cycle and with different activity statuses. The company started its work in 2009 in Alberta, Canada. The development team is globally distributed and overall 52 developers contributed to the projects. With the COVID-19 pandemic, the company experienced a change in the team dynamic and the customer demand in the healthcare offering. **Brighsquid** have actively looked into improving the developer's experience in the team. The company identifies ownership as an important aspect of developer's experience and is interested in (i) identifying how their status is being compared with other software teams, and (ii) investigating the techniques to balance developer collaboration and accountability in a team. In this study, we focus on the first goal and investigate the status of ownership within the development team.

To enable us to make a meaningful comparison for our partnered company, we aimed to perform a systematic review and or benchmark that can serve as a gold standard for comparing the status of **Brighsquid** with the current state of practice. Literature has widely acknowledged the important role of identifying ownership in large-scale software projects for improving the quality of the software product. However, there is no one source for referring to and defining ownership. After conducting an initial analysis of the literature, we discovered 28 different definitions of ownership related to code, team, module, collective ownership, organizational ownership, module ownership, issue ownership, task ownership, build ownership, requirement ownership, code authorship, and bug authorship. The terms "Ownership" and "Code

²HIPAA: Health Insurance Portability and Accountability Act of 1996

Ownership” have been frequently referenced in various studies. Code authorship is commonly understood as the process of identifying the author of a piece of software based on its source code. With the advent of issue-tracking systems, identifying authorship in a repository has become a much simpler task. Code ownership, on the other hand, is concerned with identifying the developer(s) who is responsible for maintaining a software component or code snippet over time. As personnel and code changes occur, and with the introduction of peer coding and review practices, the code author cannot necessarily be considered the sole owner of the code.

The diversity in the range of software artifacts and the terminology, as well as the difference in the offered models, motivated us to perform a systematic literature review. We planned to use the results of the systematic literature review to analyze and evaluate the status of **Brighsquad**. In particular, we are interested in answering the below research questions (RQs).

RQ1 (Literature Review): How is ownership defined and modeled in software engineering literature?

We gathered and synthesized definitions provided for ownership within state-of-the-art software engineering, along with the different attributes used to model ownership. For each paper, we collected the author’s *definition* of ownership, the *level or granularity of ownership* (file, module, or line of code, etc.), and the type of *artifact* (code, documentation, commits, etc.) that ownership was studied. We then synthesized a comprehensive list of these attributes across the different papers.

RQ2 (Literature Review): What treatment and outcomes have been used in modeling ownership in software engineering literature?

While different studies consider a variety of artifacts and their ownership (**RQ1**), each study modeled ownership differently. We have compiled and synthesized a list of outcomes such as bugs and productivity associated with each ownership model, including team and individual behaviors.

RQ3 (Literature Review): What analytics and research methods have been employed in the study of ownership in software engineering, and to what extent are these studies replicable?

To build upon existing research and establish a meaningful baseline for new methods, we seek to understand the extent and type of evaluations conducted on ownership models. In addressing **RQ3**, we summarize the *model type* (descriptive, diagnostic, predictive, or prescriptive), *nature of the project* (open source, industrial, etc.), *accessibility of data and tools*, and *reuse of the study* (whether the study replicated a previous model).

RQ4 (Replication): How does the code ownership status in Brighsquid compare with the findings of state-of-the-art studies?

We intended to evaluate the status of code ownership in Brighsquid. To this end, together with the company, we chose the most relevant studies and performed a parallel case study to three of the studies. We applied these models to 9 active projects in Brighsquid. To answer **RQ4** we provide a comparison of reported results in the original studies with our observations from Brighsquid.

In this paper, we first describe our methodology for systematic literature review in Section 2.2 and for benchmarking in Section 2.3. In Section 2.4, we present the results of our systematic literature review and answer **RQ1**, **RQ2**, and **RQ3**. In Section 2.5, we provide more insight into the status of Brighsquid’s team and code, and report the results of our ownership replication analysis in response to **RQ4**. In Section 2.6, we discuss the implications of our work, and in Section 2.7, we list potential threats to the validity of our study. Finally, we conclude our study in Section 2.8.

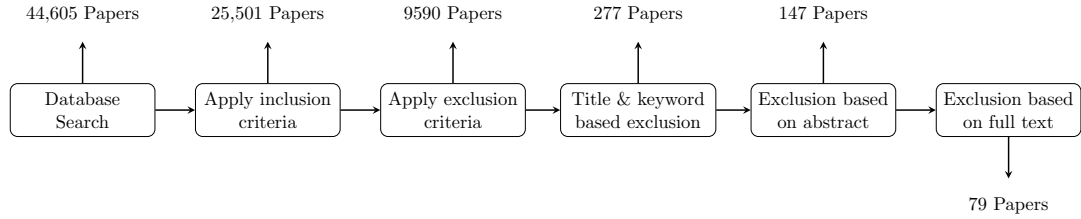


Figure 2.1: Study Selection process for our systematic literature review

2.2 Protocols Used for Systematic Literature Review

We followed the systematic literature review by Budgen and Berereton [27]. The overview of the study selection process is given in Figure 2.1. We took five main steps in this process:

STEP 1- Defining research questions; to highlight a specific area of interest the researcher defines the research questions. In this study, we have two research questions (**RQ1** and **RQ2**) as detailed in the previous section.

STEP 2- Searching for relevant papers; As the result of analyzing the research questions, researchers create searching string and find the relevant paper in the scientific database using those search string. We collected the papers for the systematic literature review among five databases commonly known and used by the research community that being *IEEE Explorer*, *Scopus*, *ACM Digital Library*, *ScienceDirect*, and *Inspec..* In Table 2.1 we listed the 17 Search strings to gather papers in the scope of this systematic mapping. From the selected databases, we collected 44,605 unique papers using our search string. Of these, there were 15,911 overlapping results between the searched databases.

Table 2.1: Search string and number of returned records within each database

| ID | Search Term | IEEE | Scopus | ACM | Science Direct | Inspec |
|----|-------------|------|--------|-----|----------------|--------|
|----|-------------|------|--------|-----|----------------|--------|

| | | | | | | |
|----|--|-----|-----|-------|-------|-----|
| 1 | “Software Engineering” & “Ownership” | 482 | 389 | 3,407 | 2,383 | 767 |
| 2 | “Software Engineering” & “Ownership” & “Code” | 84 | 94 | 2,631 | 1,702 | 358 |
| 3 | “Software Engineering” & “Ownership” & “Developer” | 51 | 64 | 1,414 | 1,208 | 67 |
| 4 | “Software Engineering” & “Ownership transfer” | 24 | 3 | 82 | 32 | 19 |
| 5 | “Software Engineering” & “Authorship” | 176 | 92 | 720 | 3,900 | 245 |
| 6 | “Software Teams” & “Ownership” | 53 | 4 | 176 | 158 | 7 |
| 7 | “Software Teams” & “Ownership transfer” | 3 | 0 | 0 | 0 | 0 |
| 8 | “Software Development” & “Ownership” | 259 | 207 | 3080 | 4,042 | 353 |
| 9 | “Software Development” & “Authorship” | 52 | 34 | 581 | 2,956 | 50 |
| 10 | “Software Engineering” & “Requirements” & “Ownership” | 59 | 42 | 2,247 | 2,040 | 80 |
| 11 | “Software Engineering” & “User Requirements” & “Ownership” | 12 | 0 | 181 | 251 | 1 |
| 12 | “Software Engineering” & “Ownership” & “Productivity” | 19 | 18 | 791 | 760 | 38 |
| 13 | “Software Development” & “Ownership” & “Productivity” | 14 | 12 | 857 | 1,436 | 28 |
| 14 | “Software Engineering” & “Ownership” & “Code” & “Productivity” | 6 | 12 | 680 | 586 | 26 |
| 15 | “Software Engineering” & “Ownership” & “Software quality” | 80 | 23 | 479 | 287 | 45 |
| 16 | “Software Engineering” & “Ownership” & “Software quality” & “Productivity” | 7 | 3 | 215 | 152 | 5 |

| | | | | | | |
|----|--|----|----|-----|-----|----|
| 17 | “Software Engineering” & “Software Development” & “Ownership” & “Software quality” | 28 | 11 | 410 | 241 | 14 |
|----|--|----|----|-----|-----|----|

STEP 3- Screening of papers, by reading the titles, keywords, and abstracts. Two annotators went through the papers and discarded papers irrelevant to the research area considering the paper title, keywords, and abstracts. When applying the inclusion criteria we ended up with 25,501 papers. Applying the exclusion criteria, from this set we took out non-English papers and 15,911 papers redundantly appeared from searching multiple databases (As detailed in Table 2.2. By that, we excluded 15,911 papers which left us with 9,590 papers.

To narrow down the search, we performed title and keyword-based exclusion and excluded papers that did not mention any of the words related to “ownership”, “authorship”, “productivity”, “quality”, “leadership”, “individual performance”, “individual vs team performance”, “team characteristics”, “team type”, or “team size” in the title or keyword. We then found 277 papers.

STEP 4- After filtering the papers based on title and keyword, for the remaining papers, by reading the abstracts, the two independent researchers scanned titles and abstracts to evaluate if the included papers match the scope of the research questions. If one of the researchers were disagreeing with the inclusion, the authors discussed the reasons and excluded papers which does not match the

Table 2.2: Inclusion and Exclusion criteria for our study

| Inclusion Criteria | | Exclusion Criteria | |
|--------------------|--|--------------------|--------------------------------|
| IC1 | Studies related to Software engineering only | EC1 | Studies not written in English |
| IC2 | Studies published in journals/conference proceedings | EC2 | Duplicated studies |
| IC3 | Studies published from 2005 to 2022 | | |

criteria.

For synthesizing the papers we made a classification scheme by considering our research questions, to answer:

- Is paper related to ownership of software artifacts?
- Is the treatment and/or outcome of the paper related to a software product or project?
- Is there an open-source repository associated with the paper or not?

Two researchers independently read the full paper abstract and categorized the papers as if the answer to each of the above questions is "Yes" or "No". We further, excluded 130 Papers that had the answer "No" for all three categories leaving us with 147 Papers to be fully read and reviewed.

During this abstract-based exclusion, we encountered a set of papers relevant to the ownership of software's *Intellectual Property (IP)* between collaborating organizations, in open source, and between different teams within the organization [28]–[30]. We also excluded papers which were related to the *software design and object ownership* discussing the code or UML diagrams [31]–[33] *cloud based ownership* [34]–[37] system level ownership [38], [39], *Non-Software document ownership* for example ownership of Wikipedia content [40]–[42]. For these papers, the answer to two of the three questions above was "No" and was not directly related to ownership of software artifacts.

Finally, we read the full text of those 147 papers and found only 79 papers that are directly related to our study. These 79 papers are the final set of papers for our study which have been studied in depth.

STEP 5- Data extraction and mapping, in this final step authors, independently go through the full body of papers and gather and synthesize information relevant

to each research question. This step is about extracting data including all the detailed information which are necessary to present and analyze our research questions.

At this stage, we recorded all the data needed for answering the research questions in detail by gathering all the ownership definitions, research questions studied in the papers, type of applied data analytics, dependent and independent variables used for modeling, empirical and evaluation setup, as well as the replication status and the availability.

By gathering all the information following the described empirical protocol, the authors synthesized and structured the information to answer the research questions. In the following section, we discuss the results of our literature review in accordance with each research question.

2.3 Protocols Used for Benchmarking and Replication

The field of software engineering has relied heavily on empirical studies and experimentation to advance its state of the art and practice. One crucial aspect of empirical research is replication, where an independent group of researchers externally reproduces the results of a study. Replication is widely regarded as a cornerstone of many disciplines, serving as an essential means of verifying and validating empirical studies [43], [44]. However, not all replication studies can be or need to be translated and discussed to verify the results of an initial study. Replications help researchers understand the impact and significance of contextual factors on the validity of empirical findings. Without proper replication, it is unclear whether study results are accidental, artifactual, or actually conform to reality [44].

In this context, we were interested in replicating the software development process within the partnered company. Our aim was not to verify the original studies but to

leverage them as an established status quo and evaluate if their results were applicable in the context of Brightsquad. Thus, our goal was to provide a comparison to assess the quality of the development processes at Brightsquad. To replicate the studies in the partnered company, we followed the steps outlined below:

- (i) Structuring the existing body of knowledge and Identifying the setup of each experimentation - We performed a systematic literature review and defined a taxonomy of what, who, and how the ownership has been measured and modeled. We identified the protocols and the measured dependent, independent, and controlled variables. These were discussed in **RQ1** and **RQ2**.
- (ii) Identifying experimentation elements - We identified the dependent, independent, and controlled variables of each model.
- (iii) Evaluating if reuse is possible - To build on the existing knowledge and to reduce divergence between our study and the existing ones, we assessed if any replication material exists. This is mapped to **RQ3** of our study. As a result of a systematic literature review, we could not identify reusable material (See Table 2.5) for analyzing our partnered company's status. Hence, in **RQ4** we took the below steps for replicating chosen studies.
- (iv) Selecting studies for replication with **Brightsquad** - Once having an overview of all the studies we discussed the context, meaningfulness, and importance of each ownership model within **Brightsquad** and prioritizing the studies for replication,
- (v) Adjusting the experimental elements for our partnered company data and code structure,
- (vi) Implementing experimental elements and performing experimentation using **Brightsquad** data,

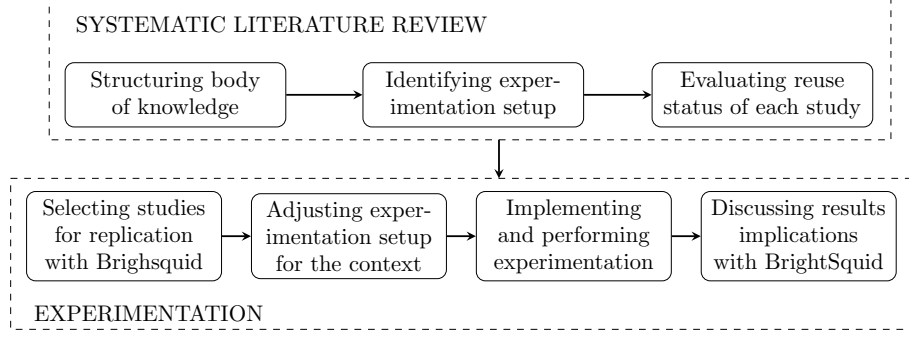


Figure 2.2: Process of replicating studies for Brighsquad

- (vii) Discussing the results in comparison with the original study and between the studies and identifying implications in collaboration with Brighsquad.

Our study methodology is summarized in Figure 2.2. Different replication types have been introduced for software engineering studies, including those discussed by Wohlin et al. [45] and Gomez et al. [44]. In our study, we performed a *close replication* by using the same research questions and following the experimental procedure as closely as possible. However, as with any non-exact replication, there were differences in factors such as the *site where the experiment is conducted*, *experimenters conducting the experiment*, *instrumentation and data*, and the *subjects conducting the experiment* [45], [46]. Using the same taxonomy, we closely followed the design chosen for the experiment and the variables measured.

2.4 Result of Systematic Literature Review (RQ1, RQ2, RQ3)

Our literature review is based on a systematic search of 79 papers, which we thoroughly examined. We included publications from 2006 to 2022, covering the period from 2005 to 2022. Figure 2.3 presents the yearly distribution of these 79 papers. We found that the highest number of papers (12 papers) was published in 2021, with the second-highest number of publications occurring in 2015.

To address **RQ2**, we conducted a thematic analysis and created a summary of the

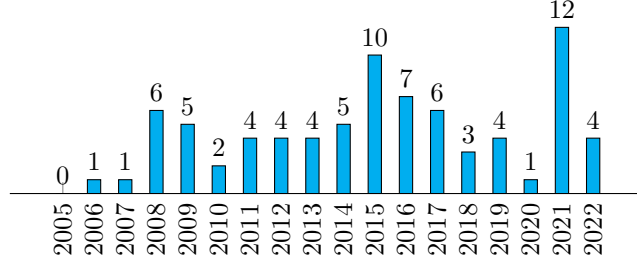


Figure 2.3: Total number of publications per year. Most papers have been published in recent years which makes this study very timely.

dependent and independent variables used to model software ownership, as well as a summary of all definitions related to the topic. In the following sections, we provide a detailed discussion of our findings in response to each research question.

2.4.1 Ownership Definitions and Models (RQ1)

The definition and categorization of ownership are largely determined by individual authors within the specific context of their studies. Consequently, the terminology used in this field is often inconsistent. To enhance clarity and comprehension, we will utilize the most widely used terminology and provide cross-references across various studies. In Software Engineering literature, the two primary types of ownership are Psychological Ownership, which pertains to an individual’s sense of ownership, and Corporal Ownership, which refers to an artifact’s development history:

Psychological ownership refers to the feeling of ownership that a person may develop toward an owned entity in the project [8].

Corporeal ownership refers to the development history of an artifact often by referring to the git history.

In the software engineering literature, the primary focus is on various aspects of Corporal ownership. According to Bird et al. [6], ownership refers to a general term used to determine whether a particular software component has a responsible developer or not. The ownership of a software artifact is subject to change based on

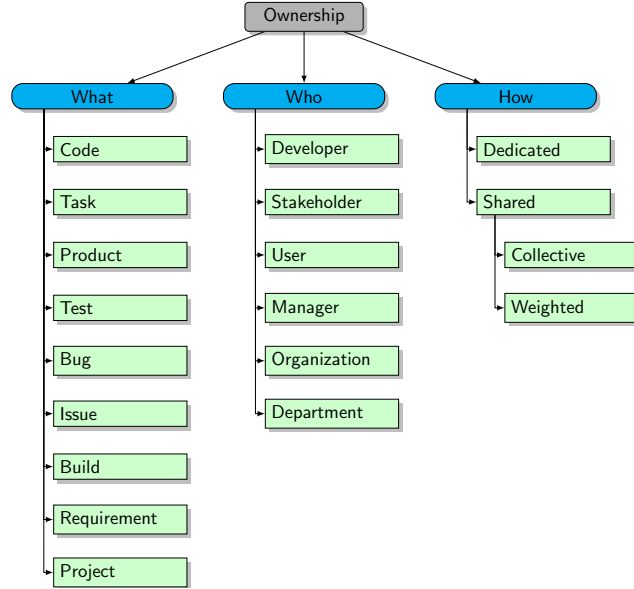


Figure 2.4: Taxonomy of the nature (What?), the owner (who?), and the degree of ownership (How?) as the result of our systematic review.

the strategies and logistics implemented by a software team. It is commonly assumed in such studies that developers have the freedom to modify other teammates' solutions whenever necessary [47], [48]. Consequently, the ownership of a software artifact is often shared among the developers. However, some studies focus on identifying an individual responsible owner for a particular artifact, while others discuss collective responsibility among team members. In this study, we present our findings about ownership from three different aspects to address **RQ1**.

Type of artifact that its ownership is being discussed within a software team,

Owner of the artifact whom their role in the maintenance of the artifact is being discussed,

Degree of ownership which indicates the extent of right and responsibility of the owner as well as the responsibility of team members in maintaining an artifact.

In the field of software engineering, some authors have used the terms authorship and ownership interchangeably [49] [50]. For example, Muller et al. [50] use the term

code ownership to refer to the authorship of a particular software artifact; “The term code ownership is used in software engineering to describe who authored a certain piece of software.” However, in our systematic review, we distinguish between authorship and ownership as two distinct yet related concepts. Authorship pertains to identifying the author of an artifact, such as a line of code. Bogomolov et al. [51] formulated the authorship problem as “given a piece of code and a predefined set of authors, to attribute this piece to one of these authors, or judge that it was written by someone else.”

In contrast, ownership involves identifying the member(s) responsible for maintaining a software artifact, whether or not they originally wrote it [52]. The collaborative nature of software maintenance and dynamic code changes make it difficult to identify a single developer as the author of any artifact. Rahman and Devanbu differentiated between the terms authorship and ownership [53]. They considered any developer with a contribution to a code fragment as an author and referred to the developer with the highest contribution as the owner of a code fragment. In our systematic literature review, we also differentiate between these two terms, recognizing that ownership at all levels can be shared or individual.

Type of Artifact (What?)

Software artifacts are tangible products that result from development activities within software teams. Through our systematic literature review, we identified nine different artifacts whose ownership has been the subject of one or more studies. Table 2.3 provides a condensed definition of the ownership of each artifact, as a result of our detailed synthesis in this systematic literature review.

Among all the artifacts, “**Code**” ownership has been studied most frequently. Hattori and Lanza [7] discussed code ownership as identifying the developer who owns an artifact of a software system “by measuring who has accumulated more knowledge of each artifact”. Nagappan et al. [54] studied the impact of organizational

Table 2.3: Types and definition of corporal ownership in software engineering state of the art (**RQ1**).

| Artifact | Definition | Study |
|-------------|--|--|
| Code | Code author is the developer who writes a piece of source code. Code owner is defined by measuring who has accumulated more knowledge of each software component (e.g. module, file, class, binary). | S1, S3, S4, S5, S6, S7, S8, S10, S11, S12, S13, S15, S16, S17, S18, S19, S20, S21, S23, S30, S35, S36, S37, S39, S41, S42, S43, S44, S45, S49, S57, S62, S70 |
| Product | Product owner is the person who is responsible for managing the Product Backlog so as to maximize the value of the project. | S2, S48, S59, S69, S78 |
| Task | Task owner is the developer who accepts the responsibility of doing a task from a team’s task list. | S14, S52, S64 |
| Issue | Issue Owner is the developer who resolves an identified issue in an artifact and he may not be the same person who does work previously. | S9, S73 |
| Project | Project owners are all members of a team who are responsible for the work completion of a project. | S46 |
| Test | Test owner is any distinct developer that owns at least one test in the test suite. | S25 |
| Bug | Bug owner is the developer who has written the code that caused the bug. | S24 |
| Build | Build ownership is the proportion of the developers who are responsible for build maintenance. | S28 |
| Requirement | Requirement owner is a person who provides a requirement/requirements for a system. | S38 |

factors on code quality. Among the eight organizational metrics, they invested in the depth of Master Ownership (DMO). They defined the developer with more than 75% edits in a binary file as the owner and considered their organizational level as the DMO. They also defined the Level of Organizational Code Ownership (OCO) as the percentage of edits from the organization that contains the owner of the binary file. The authors also introduced Overall Organization Ownership (OOW) as the ratio of the percentage of developers at the DMO level editing a binary file to the total number of engineers editing that file. Meng et al. [55] discussed line-level code authorship by introducing repository graphs, structural authorship, and weighted authorship. They use Structural authorship and weighted authorship to extract the development history of a line of code and approximate its authorship. ”Structural authorship is a subgraph of the repository graph. The nodes consist of the commits that changed

that line. Development dependencies between the subset commits form the edges.” Weighted authorship ”is a vector of author contribution weights derived from the structural authorship of the line. The weight of an author is defined by a code change measure between commits”. Businge et al. [49] used the amount of code change as defined by earlier studies [56] to estimate the code ownership for an application. In the XP methodology, the ideal code ownership scenario is defined as ”any developer can refactor any area of code in an application as long as it continues to meet the contract defined by interfaces and unit tests” [56]. As the opposition, Judy et al. [18] considered the notion of code ownership stating that a single accountable authority on such is impractical. Hence, they introduce *product owner* as the individual (one person and not a team) that is accountable for achieving business objectives. Overall, 33 Papers specifically referred to the code ownership artifacts.

”**Product**” is the second most frequently appeared artifact in our literature review and its ownership has been discussed in the literature though only four studies specifically mentioned it. The product owner is a popular term for agile projects. Judy et al. stated that the Product Owner is typically one person and responsible for managing the product backlog, which contains an emerging set of requirements. They also suggest that the success of the product depends largely on the decisions made by the product owner. Additionally, Shastri et al. note that the product owner is often the customer representative.

Judy et al. [18] state that Product Owner is typically one person and responsible for managing the product backlog, which contains an emerging set of requirements. The success of a product beyond the reliability, supportability, scalability, and time to market of its implementation entirely depends on the decision of the product owner [18]. Furthermore, Shastri et al. [57] note that the Product Owner is also commonly regarded as the customer representative. ”**Project**” ownership occurs when all the members of a team take the responsibility to complete the project. It appears three times in our literature review whereas one paper defined it clearly. In

[58], authors stated that in project ownership all the team members are responsible for work completion and they also share benefits among them. Burga et al. [10] discussed that both project managers and project have a relationship within the project governance structure. **"Task"** ownership was specially studied for agile projects. In a team, team members take ownership of tasks from the task board by taking issues and risks [19]. Datta [59] examined factors that influence task ownership in Android development by introducing eight different variables. **"Requirement"** is a newer concept among all artifacts of ownership. Hadad et al. [60] studied requirement ownership by identifying the ownership of requirements of a system. They studied different five types of such ownership where only developers are the owner of each requirement, or where clients provide requirements of a system, where users are the author of a system requirement, or where ownership is collaborating and engineer, client, and user are equally responsible for every requirement, or where ownership is diffused and there is no one person for a requirement.

"Test" ownership has also been discussed by Herzig and Nagappan [61] and has later been referred to by studies focusing on code quality. Within a software team, a test owner is a developer who owns at least one test case from a test suite [61]. It appears only once [61] in our literature review where authors showed that organizational structure impacts test case quality. They defined test case quality through test effectiveness and test execution reliability. They calculated the Spearman rank correlations between sixteen organizational metrics and measures of test suite effectiveness and test suite reliability and suggest that test suites whose owners are distributed over multiple organization subgroups with long communication paths are negatively correlated with quality. **"Issue"** ownership artifact is discussed in two papers in our literature review. Caglayan et al. [62] discussed issue ownership by identifying the developer who resolves the issue. The authors showed that a small group of developers tends to take ownership of a large portion of new issues especially when the active issue count is relatively high in the software development life cycle. Issue ownership

can be individual or teamwork [63] and the studies differentiate between the developer assigned to the issue and the one who resolves the issue [64]. Along the same line, **"Bug"** ownership studied by Zhu et al. [65]. The developer who submitted the bug-introducing commit is known as the bug owner [65]. A bug can be fixed by the developer who introduced it or by a different developer. Zhu et al. [65] found that bug-fixing time by the bug owner is much shorter than that of other developers. They also found that the owner's bug fixing commits are larger and have a different pattern in comparison to non-owners.

"Build" ownership is mainly discussed to maintain and change build systems. In their study, McIntosh et al. [66] proposed a definition of build ownership based on the proportion of developers responsible for maintaining a build system. The authors also identified two distinct styles of build ownership. The first style, referred to as "concentrated," involves a small dedicated team responsible for build system maintenance. The second style, called "dispersed," is characterized by the majority of developers contributing code to the build system. Shridhar et al. [67] studied six different build change categories and three different types of ownership styles.

Type of Owner (Who?)

The majority of the studies in our literature review (82%) solely search for identifying a set of "developers" (D_i where $i \geq 1$) as the owners of an artifact. Here, "developer" is the general term used for a software team member. A few studies (14 out of 79) distinguished between the team members, either considering their organizational status or considering their experience in the team.

Nagappan et al. [54] investigated the impact of organizational structure on software quality. In particular, they consider the organizational level of engineers in **Microsoft** for Windows Vista. They consider engineers, managers, and sub-managers within their analysis and introduce a number of metrics to capture the organizational structure (see Appendix I, Table 4.1, for further details). Bird et al. [6] examined the

ownership effect on code quality. While ownership metrics are discussed here from the developers' perspective, project managers are also responsible for checking the code of a developer with less relevant experience. In [68], authors replicated [6] by introducing some more ownership metrics, which are about the organizational level of developer and manager.

Thongtanunam et al. [52] discussed reviewing activity in terms of code ownership. In particular, they introduce some control variables using the code review to identify code ownership. In this study, they mainly extend the code ownership into review activity and consider the reviewer instead of the developer within their research. In [61], test ownership is discussed from the test engineer's perspective. They showed organizational structure impacts test quality by introducing some organizational metrics. Meneely et al. [69] indicate that team characteristics like team size and team expansion rate affect software quality. They did their analysis at the department level. Hadad et al. [60] introduced requirement authorship and discussed five authorship types. They consider users, stakeholders, and developers as the owner of software requirements in their research. In Figure 2.4 we abstracted these roles into six major entities by referring to the common roles within software teams.

Degree of ownership (How?)

The majority (60 out of 79) of the studies acknowledge and formulated the authorship issue as a problem of assigning a degree of responsibility to the different team members who edited a code snippet. For a software team consisting of D individuals represented as $Developer = \{dev(1), dev(2), \dots, dev(i)\}$ and a set of artifacts developed by the team $Artifact = \{art(1), art(2), \dots, art(j)\}$ our literature review and synthesis resulted in three general ownership models.

With the introduction of version control systems, software engineering literature gained the ability to study the concept of ownership and model fuzzy and shared ownership between multiple developers. The most commonly accepted ownership

model is the **”Weighted”** model, which was followed by 31 out of the 79 studies in our systematic literature review. In this model, the ownership of an artifact is determined based on factors such as familiarity or effort spent on the artifact. The ownership of each artifact is represented as a vector of owners, where each owner is assigned a weight that corresponds to their contribution to the artifact’s changes. For example, Meng et al. [55] defined the weighted ownership of a code snippet as a vector of developers’ contribution weights using the churn per commit. Where the total amount of contributions made by a developer d , to an artifact a is $Contribution(i, a)$;

$$Weight(d, a) = \frac{Contribution(d, a)}{\sum_{i=1}^{i=D} Contribution(i, a)} \quad (2.1)$$

and

$$\sum_{i=1}^{i=D} Weight(a, i) = 1 \quad (2.2)$$

In this model, despite the unit which is being used for measuring the developer’s contribution to an artifact (e.g., number of lines of code, churn, number of commits), the weight is the relative contribution of each developer to the rest of the team.

The literature in particular refers to two special cases of this model. First, where there is only one developer is being identified as the owner of an artifact (we commonly refer to that as the authorship problem). Second, where every developer in the team is responsible for all the artifacts despite their degree of involvement. The problem of finding one developer as the responsible individual for writing and maintaining a line of code is commonly known as the authorship problem in software engineering. **”Dedicated”** Ownership searches for one responsible developer for an artifact and identify only one individual as the owner of an artifact. In this case, in Equation (1) for developer $dev(d)$ which is the author of an artifact;

$$Weight(a, i) = \begin{cases} 1, & \text{if } i = d \\ 0, & \text{otherwise} \end{cases} \quad (2.3)$$

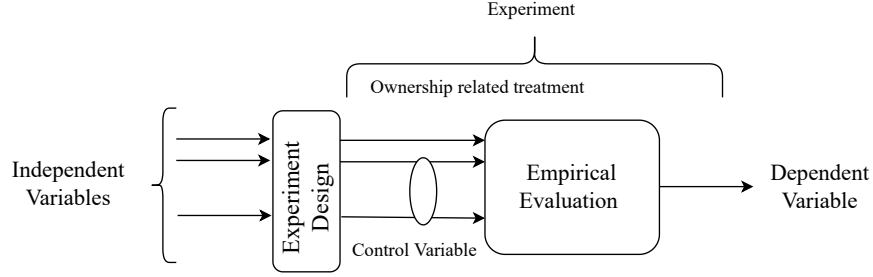


Figure 2.5: Treatment-outcome relation as we studied in this SLR. The process and the diagram are adopted from the work of Wohlin et al. [45].

”**Collective**” ownership is focused on getting the tasks (such as a bug fix) done by leveraging the expertise of the entire team. Teams with a high level of collective ownership have all the team members taking ownership of the software product. The assumption is that every team member has a minimum degree of expertise and knowledge of other’s work through collaboration, coordination, or in consideration of their organizational role. Maruping et al. [70] use the notion of collective ownership. In this case for any $art(a)$ and $dev(i)$;

$$Weight(a, i) \neq 0 \quad (2.4)$$

”Dedicated” and ”Collective” ownership models are the two extremes of the ownership range where either an individual (dedicated) or the whole team (collective) is responsible for an artifact.

2.4.2 Ownership Modeling Treatments and Outcomes (RQ2)

Ownership models in software engineering use different attributes and artifacts. We followed and referred to the general experimentation process as provided by Wohlin et al. [45] and outlined in Figure 2.5. As the result of the systematic review, we identified 18 unique dependent, 73 independent, and 18 control variables. These variables are detailed in Appendix II and Table 4.1. In Table 2.4, we presented only those dependent, independent, and control variables used more than once in our literature review.

Table 2.4: Independent, Dependent, and controlled variables used “more than once” in modeling ownership studies.

| Independent Variable | Study |
|---|---|
| Number of developers changing a software component | S3, S6, S7, S9, S15, S16, S18, S19, S26 |
| Major contributor (developer with majority of commits to a component) | S7, S15, S16, S19, S23, S29 |
| Minor contributor (developer with minority of commits to a component) | S7, S15, S16, S19, S23, S29 |
| Highest % of commits a developer made to a software component | S7, S15, S16, S19, S23 |
| Code size | S19, S43, S54, S65 |
| Code Churn | S3, S4, S6, S19, S77 |
| Number of commits | S4, S11, S18, S77 |
| Code complexity | S3, S6, S77 |
| Developer experience | S8, S54 |
| Number of managers | S16, S25 |
| Dependent Variable | Study |
| Number of post-release failures | S3, S6, S7, S9, S11, S19 |
| Number of bugs | S13, S15, S16, S18, S23, S54 |
| Number of pre-release failures | S7, S11, S21, S29 |
| Controlled Variable | Study |
| Code complexity | S7, S11, S15, S23, S29, S72 |
| Code size | S7, S11, S15, S21, S29 |
| Project size | S1, S8, S23, S62 |
| Code churn | S7, S11, S21 |
| Developer experience | S1, S72 |
| Number of developers | S8, S21 |
| Number of developers participating in the review process (reviewers) | S14, S21 |

We identified 73 identical *independent variables*. The most popular independent variable in our literature review is the NUMBER OF DEVELOPERS who changed a software component. The NUMBER OF DEVELOPERS has been used as the independent variable in nine papers and in relevance to the ownership of code and issues. The other popular independent variable within our literature review is the number of MAJOR AND MINOR CONTRIBUTORS. These two were discussed in accordance with each other and within six papers. Here, a MAJOR CONTRIBUTOR is a contributor who made more than a predefined threshold of the changes (commits) to a component. Similarly, a MINOR CONTRIBUTOR is a contributor who made less than a predefined threshold of the changes (commits) to a component, and ownership is the highest value of the ratio of contributions performed by all developers. CODE CHURN, CODE SIZE, and the NUMBER OF COMMITS have been discussed in five to three studies. 63 independent variables are unique to one study within our literature review which are mostly relevant to test ownership, code ownership, and product ownership. These independent variables have never been chosen by another study within the scope of our studied literature. Details of all these variables are in Appendix II.

We identified a total of 18 *dependent variables*, which are detailed in Appendix II. The two most frequently appearing dependent variables in our literature review were the NUMBER OF BUGS and the NUMBER OF POST-RELEASE FAILURES, each appearing in six different studies. This was followed by the NUMBER OF PRE-RELEASE FAILURES, which was used as a dependent variable by four studies. The remaining 15 dependent variables were used only in one study each. More than one-fourth of the studies in our literature review (23% of the studies in our literature review) chose a form of code quality metric as a dependent variable, resulting in the identification of 10 different metrics of code quality as dependent variables in code ownership studies.

We also identified 18 distinct *controlled variables*. The most popular controlled variable was code complexity, chosen by six different studies. SIZE, specifically CODE SIZE, was chosen by five studies, and PROJECT SIZE was selected by four different

studies as a controlled variable. Additionally, CODE CHURN, DEVELOPER’S EXPERIENCE, NUMBER OF DEVELOPERS, and NUMBER OF CODE REVIEWERS were used as controlled variables more than once. Our analysis revealed a total of 18 controlled variables, with 11 of them appearing in only one study within our literature (detailed in Appendix I).

2.4.3 Analytical Model, Replicability, and Evaluation setup (RQ3)

To summarize the papers included in our systematic review, we examined the analytical models used in each study and categorized them according to the four models introduced by Kaisler et al. [71]:

Descriptive Analytics is the most basic and commonly used model. It analyzes existing data statistically to identify patterns and gain an understanding of the overall state of the data.

Diagnostic Analytics focuses on identifying the root causes of issues or anomalies within existing data. It answers the ”why” question by providing deep insights into why certain events occurred.

Predictive Analytics utilizes machine learning techniques to forecast likely future outcomes based on existing data. This type of analysis helps to anticipate potential outcomes and enables informed decision-making.

Prescriptive Analytics combines descriptive and predictive models to provide guidance on the best course of action for achieving optimal outcomes. This model is focused on the decision-making process.

We applied Wendler’s research method schema [72] to classify the type of research evaluation used in each study. The classification includes case study, survey, interview, concept development, literature review, mixed approach, and others. Additionally, we

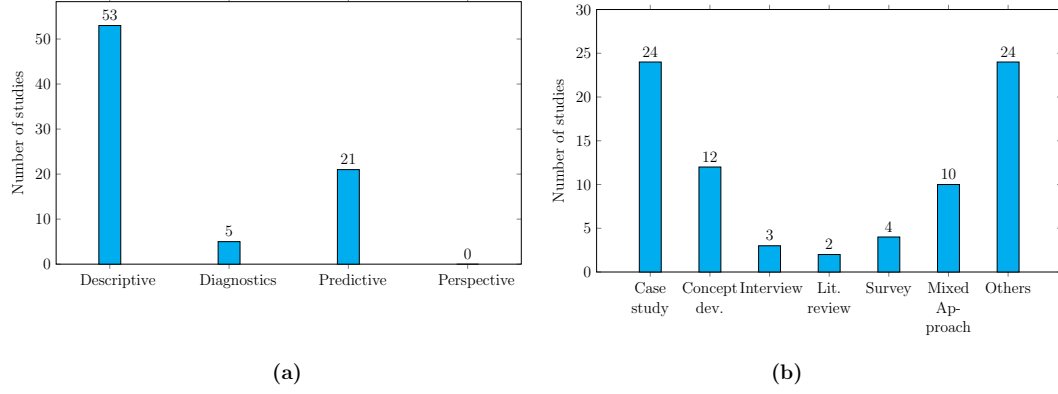


Figure 2.6: Distribution of paper based on (a) Type of analytics and (b) Research Method

examined the availability of data and code for each study to facilitate replication, as per **RQ4**. The summary of our findings is presented in Figure 2.6-(a) and Figure 2.6-(b), where Figure 2.6-(a) shows the number of papers in each analytic category, while Figure 2.6-(b) shows the number of papers in each research method category.

Descriptive analytics was the most commonly used analytic model, used in 67% of the studies, while we did not identify any study as prescriptive. Predictive analytics was used in 21 studies (26.6%), and four studies (6.3%) used diagnostic analytics. Figure 2.6-(a) provides a detailed breakdown of the analytics used in each study.

Regarding the research method used, case study was the most commonly used approach, with 24 studies (30.3%) following this method. Concept development was used in 12 studies (16.0%), followed by interview-based and survey-based studies,

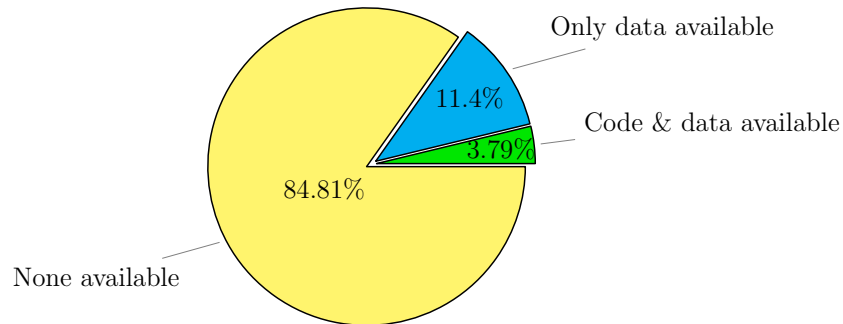


Figure 2.7: Percentage of papers which their code and data are available

which were used in three (4.0%) and four (5.3%) studies, respectively. Only two studies were based on literature reviews. Ten studies used a mixed approach, such as Drury-Grogan et al.[73], who performed both a case study and an interview to identify decision characteristics. Finally, 24 papers could not be classified into any of these research method categories and were tagged as "others." Figure 2.6-(b) provides a detailed breakdown of the research method used in each study.

We also were interested to understand the applicability status of these studies. We manually checked if the data or the code are available for each study at the time of writing this paper (Figure 2.7). We found that only 9 studies (11.4%) within our literature review provided their dataset link. Among them, 3 studies (3.79%) also provided source code. Moreover, we observed that 32 studies considered open-source data for their research. Table 2.5 summarizes the overview of these studies.

Table 2.5: Data availability within our systematic literature review. ✕ shows the unavailability of the data while ✓ represents data availability.

| Paper | Data | Dataset Description |
|-------|------|---|
| S3 | ✕ | 3,404 binaries exceeding 50 Million lines of code of Windows Vista. |
| S4 | ✕ | 2,429 changes performed by four developers from one commercial project. |
| S5 | ✕ | Organizational database (POST) to get the information of 29 to 252 developers having from 12 to 482 followers and the code changes they made over a period of seven years. |
| S6 | ✕ | 511 lines of code for windows Vista over seven teams in different geographical locations. |
| S7 | ✕ | Commit histories and software failures of two Microsoft projects, Windows Vista and Windows 7. |
| S8 | ✕ | Four open source project Apache, Nautilus, Evolution and Gimp. |
| S9 | ✕ | A large-scale enterprise software with more than twenty years of development history, and (ii) a smart phone Android operating system developed by Google and distributed as open source. |
| S10 | ✕ | 10 open source software products written in Java. |
| S11 | ✕ | Source code repositories, bugs, and the locations of developers of Firefox and Eclipse projects. |

| | | |
|-----|----------------|--|
| S12 | × | Student-submitted solutions to programming assignments written in C from RMIT University. |
| S13 | × | Five open source projects Dyninst , Httpd , GCC , Linux , GIMP . |
| S14 | ✓ ³ | Android code review data from Gerrit-Miner . |
| S15 | ✓ ⁴ | Seven open source projects Apache Ant , Camel , Log4J , Lucene , Eclipse JDT Core , PDE UI , and Equinox . |
| S16 | × | Four major Microsoft products being Office , Office365 , Exchange , and Windows repository. |
| S17 | × | Change history from version control system for VISUAL STUDIO 2013 . |
| S18 | × | Repository of six open source Apache projects |
| S19 | ✓ ⁵ | Repository of seven open source project |
| S20 | | Four open source software Ant , Gremon , Struts2 , Tomcat . |
| S21 | × | Code review dataset of two open source project Qt and Open Stack . |
| S23 | × | 45,000 open source Android -based repositories hosted on GitHub . |
| S24 | ✓ ⁶ | SStuBs dataset which contains 10,231 bug fixes from the top 100 Java Maven projects. |
| S25 | × | organizational metrics datasets provided by the CODEMINE project. |
| S26 | × | Code logs, bugs, and historical records from the Human Resources department from Cisco projects. |
| S27 | ✓ ⁷ | code and bugs of five open source project Angular.JS , Ansible , Jenkins , JQuery , and Rails . |
| S28 | × | Build log of 18 Apache and Eclips open-source projects over a period of fourteen months. |
| S29 | × | Code and reiews of a large scale commercial project. |
| S30 | × | 16 years entire history of the Apache Ant system. |
| S31 | ✓ ⁸ | Pull Requests of 246 open source projects hosted on GitHub . |
| S33 | × | 41,140 open source code file changes from Eclipse and and 6,548 function changes for a proprietary software project both between January 2001 and September 2013. |

³<https://github.com/mmukadam/gerrit-miner.git>

⁴<http://bug.inf.usi.ch/>

⁵<https://se.labri.fr/data/articles/IST-2014/>

⁶<https://anonymous.4open.science/r/344cf208-ea32-49f4-90fe-59bdb6e5d7fe/>

⁷<https://se.labri.fr/a/FSE15-foucault/>

⁸<https://github.com/yuyue/pullreq.ci>

| | | |
|-----|-----------------|--|
| S34 | × | 132 versions of PLAY released between 2008 and 2013 including 14 evolutionary and 118 maintenance releases. |
| S35 | ✓ ⁹ | 270,000 commits of IntelliJ IDEA Community Edition project authored by 500 developers between 2004 to 2020. |
| S36 | × | Source code of nine different authors from the Codeisplanet website. |
| S37 | × | Four open source systems namely ArgoUML , Mozilla , Samba , and Squid . |
| S39 | × | 150 collaboratively written C++ files for 106 programmers active on GitHub . |
| S40 | × | GNU C compiler GCC version 3.3.2 open source project. |
| S41 | × | 1,000 programmers' data, collected from Google Code Jam (GCJ) . |
| S43 | × | Six Apache projects naming Activemq , Aries , Carbondata , Cassandra , Derby , Mahout . |
| S44 | × | 200,000 source code files written by 898 developers from open source projects on GitHub . |
| S45 | × | Several open source projects, including Apache HTTP Server , Dyninst , GCC and other projects from GitHub . |
| S50 | × | SDT software project user requests over six month period. |
| S51 | × | 200 open source projects belonging to Android , Apache , and Eclipse . |
| S53 | × | Development history of four releases of a company. |
| S54 | × | Commit history and issue from 12 GIT repository. |
| S57 | × | Google Code Jam (GCJ) development log between 2008 to 2016 and 1,987 repositories hosted on GitHub . |
| S62 | ✓ ¹⁰ | 66 releases of 118 open source projects on GitHub . |
| S63 | × | Developer's communication data for three open source projects naming Gaim , eGroupWare , and Compiere . |
| S66 | × | Jira issues from four open source Java projects: Hibernate , JBoss , Mule and Spring . |
| S68 | ✓ ¹¹ | 200 most forked Java projects from GitHub . |
| S70 | × | contest code from 2008 to 2016 of Google code Jam |
| S72 | × | 1.2 million commits and more than 25,000 developers of multiple Open Source as well as a proprietary software project |

⁹<https://github.com/JetBrains-Research/authorship-detection>

¹⁰<https://github.com/torvalds/linux>

¹¹https://github.com/acapiluppi/oometrics_developers

| | | |
|-----|---|--|
| S75 | × | Commits and issues of GitHub repository |
| S77 | × | Source code, version and Jira issues of a commercial project |

2.5 Benchmarking State of the Art Model in Brightsquad

We discussed the result of our literature review with the partner company (Brightsquad) to identify the most relevant literature. The intent was to compare the status quo of "code ownership" at Brightsquad in comparison to the other companies. We used the taxonomy we extracted in **RQ1** to identify what, who, and how the ownership of code has changed at Brightsquad, in comparison to other companies. As the result, we chose three studies to replicate and compare with the status of code ownership at Brightsquad:

Study by Bird et al. [6]: This widely-cited study from 2011 developed a model for code ownership based on proprietary software at Microsoft, analyzing both code and component ownership. Many studies have built upon these findings [S7]. The authors posed three main research questions (listed in Table 2.6), and we answered two of these questions for all nine Brightsquad projects.

Study by Greiler et al. [68]: In 2015, Greiler et al. replicated and extended the model from Bird et al.'s study, using file and directory-level analysis and introducing new thresholds for identifying owners. This work was of interest to our partnered collaborators since it is one of the few studies on non-open source projects [S16].

Study by Foucault et al. [74]: While focused on open-source software projects, this paper provides a comprehensive overview of the metrics used in the literature. We intended to apply the relevant aspects of the model to Brightsquad [S19].

Table 2.6: Research Questions addressed in the three replicated studies

| Study | Research Question | Answered? |
|----------------------|--|-----------|
| Bird et al. [6] | RQ_{S71} : Are Higher levels of ownership associated with less defects? | Yes |
| | RQ_{S72} : Is there a negative effect when a software entity is developed by many people with low ownership? | Yes |
| | RQ_{S73} : Are these effects related to the development process used? | No |
| Greiler et al. [68] | RQ_{S161} : Are ownership metrics indicative for code quality for other software systems than for windows? | Yes |
| | RQ_{S162} : For which levels of granularity (directory and source file) do ownership metrics correlate with code quality measured by the number of fixed bugs? | Yes |
| | RQ_{S163} : Can ownership metrics be used to build classification models to identify defective directories and defective source files? | Yes |
| | RQ_{S164} : What are the reasons for the lack of ownership? | No |
| Foucault et al. [74] | RQ_{S191} : Does code ownership measured via the code ownership (CO) metrics, most valued owner (MVO), Minor, and Major, have a relationship with software modules quality, measured with their number of post-release bugs? | Yes |
| | RQ_{S192} : If so, do these metrics provide an added value for predicting the number of post-release bugs of a software module? | Yes |

In Table 2.6, we have listed all the research questions for the three studies under consideration. While these studies have answered nine different research questions, there is a significant overlap between the tested hypotheses. However, the empirical methods used in these studies, such as the chosen dependent, independent, and controlled variables, differ. After consulting with our collaborators, we have decided to exclude two of the research questions. Specifically, we have excluded the third question of Bird et al. [6] (RQ_{S73} in Table 2.6), which investigates the impact of the development process on the related outcome through an interview. A similar approach was taken by Greiler et al. [68] using qualitative analysis to understand the reasons for the lack of ownership within a team.

To answer each of the remaining research questions, we used the results of our systematic literature review, along with our extracted lists of dependent, independent, and controlled variables for each study. We then gathered relevant data and metrics

to provide answers to these research questions.

2.5.1 Hypothesis and Research Methodology in Original Studies

Bird et al. [6] [S7] built their work based on four hypotheses, two of them associating the ownership of a software component with the number of failures:

Hypothesis 1: Software components with many MINOR contributors will have more failures than software components that have fewer.

Hypothesis 2: Software components with a high level of OWNERSHIP will have fewer failures than components with lower top ownership levels.

Hypothesis 3: MINOR contributors to components will be MAJOR contributors to other components that are related through dependency relationships.

Hypothesis 4: Removal of MINOR contribution information from defect prediction techniques will decrease performance dramatically.

Analogue to this study, Greiler et al. [68][S16] tested **the same hypothesis** with a few differences for the same corporation:

- *Changing the granularity* - Looking into file level and code directory level instead of component ownership. Source directories contain those source code files which has the same enclosing path as defined by the original study.
- *Changing the threshold for identifying minor ownership* - authors used less than 50% for identifying MINOR owners.
- *Introducing MINIMAL ownership* - MINIMAL owners are the developers having less than 20% of ownership.
- *Consider strict ownership* - authors separately analyzed the files that have been edited by only one person. They considered these files as having strict ownership.

Foucault et al . [74] [S19] as well investigated the same four hypotheses from Bird et al. [6] with a number of differences:

- They tested the hypothesis in six open-source projects.
- They only invested in the number of post-release bugs as the dependent variable.
- They manually identified software modules using the knowledge of the research team. A software module could be a file or a directory.
- Considering TOUCHE (the number of files touched by a developer) in addition to CHURN.
- Giving a new name to the OWNERSHIP metric and calls it MOST VALUED OWNER (MVO).

We summarized all the dependent and control variables explained in Table 2.7 and independent variables in Table 2.8 and separated each of these studies.

Table 2.7: What, who, and How ownership was discussed in the selected studies for replication and the dependent, and control variables.

| ID | What? | Who? | How? | Dependent variable | Control variable |
|-----|--------------------------------|----------------------------------|-------------------|---|--|
| S7 | Binary files (Code, Component) | Developer, Manager | Weighted | Number of pre-release failures, Number of post-release failures | Code size, Code churn, Code complexity |
| S16 | Binary files (Code, Component) | Developer, Manager, Organization | Collective shared | Number of bugs | N/A |
| S19 | Directories (Code, Component) | Developer) | Shared | Number of post-release bugs | N/A |

Table 2.8: Independent variable in the selected studies for replication.

| Independent Variables | | |
|-----------------------|-----------------|---|
| ID | Metric name | Definition |
| S7 | Major | The number of developers who made more than 5% commits to a file |
| | Minor | The number of developers who made less than 5% commits to a file. |
| | Total | The total number of developers who contributed to a file. |
| | Ownership | Proportion of commits for the highest contributor to a file. |
| S16 | Minors | The number of developers who made less than 50% commits. |
| | Minimals | The number of developers who made less than 20% commits. |
| | Contributors | Total number of contributors. |
| | Ownership | Proportion of commits for the highest contributor. This definition is identical to the one provided by Bird et al. [56] [S7] |
| | Avgownership | Average ownership values for all files in a directory: $(\sum \text{fileownership})/(\# \text{files})$. |
| | Ownershipperdir | % of commits of the highest contributor considering all files in a directory: $(\sum \text{ofmaincontributorcommits})/(\# \text{allcommits})$. |
| | Minownerdir | % of commits of the lowest contributor considering all files in a directory: $(\sum \text{ofcommitsofthelowestcontributor})/(\# \text{allcommits})$. |
| | Avgcontributors | Average of distinct contributors among all files in a directory: $(\sum \text{ofdistinctcontributorsperdirectory})/(\text{files})$ |
| | Pcminors | % of contributors among all contributors with less than 50% commits for all directory files: $(\sum \text{ofdistinctminors})/(\# \text{contributors})$. |
| | Pcminimals | % of contributors among all contributors with less than 20% commits across all directory files: $(\sum \text{ofdistinctminimals})/(\# \text{contributors})$. |
| | Pcmajors | % of contributors among all contributors with more than or 50% commits across all directory files: in a directory: $(\sum \text{ofdistinctcontributorswithmorethan50\%changes})/(\# \text{contributors})$. |
| | Avgminimal | Average minimals in a directory: $(\sum \text{ofminimalsperfile}/\# \text{files})$. |
| | Avgminors | Average minors in a directory: $(\sum \text{ofminorsperfile}/\# \text{files})$. |
| | Minownedfile | The ownership value of the file with the lowest ownership value. |
| | Weakowneds | Number of files in a directory that have an ownership value of less than 50%: $(\# \text{of files with } < 50\% \text{ownership})/\# \text{files}$. |
| S19 | Touches | The number of files touched by a developer. |

The empirical methods used to investigate the hypotheses in both Bird et al.[6] and Foucault et al.[74] are largely the same, despite differences in subjects and some metrics. In the first step (Step 1 in Figure 2.9), they analyzed the correlation between code attributes and ownership metrics. Subsequently (Step 2), they developed a mul-

multiple linear regression model to identify the relationship between ownership metrics and the number of failures while controlling for source code characteristics. To test the four hypotheses and report on their results, they compared the variance in failure (also known as \mathbb{R}^2) (Step 3), and assessed goodness of fit using F-tests (Step 4).

Greiler et al. [68] took their first step similar to the other two studies (Step 1) and started by looking into the correlation between variables. Then they take a step further (Step 2 of Figure 2.9) and built a Random Forest model to predict source code files and directories associated with at least one bug. Then (Step 3) they performed feature selection using Principal Component Analysis (PCA) to reduce the number of orthogonal dimensions. In Step 4, they measured the variable importance for every individual ownership metric by calculating the area under the ROC diagram. They complemented the quantitative analysis by performing interviews and surveys and the qualitative description of the developer’s perception (Step 5).

2.5.2 Empirical Data from Brightsquad

At the time of writing this paper, Brightsquad has a total of 39 projects, each starting at a different time between 2012 to 2023. A total of 52 developers worked across all these projects. Also, 13,330 issues exist on the issue tracking system of the company. In collaboration with our industry partner, we only considered the projects active after March 2018 for our analysis. As a result, we limited our analysis to nine projects at Brightsquad as others have been idle at that time. Table 2.9 summarize the project status based on the number of commits, number of bug fixes, and the lines of code. All the project source codes are hosted on GitHub, and the company uses Jira as the issue tracking system. The issues are being traced to the source code using automated integration tools. Below, we describe the data gathered for replicating the selected research questions from the studies, as listed in Table 2.6. We gathered these data through GitHub and Jira APIs. An NDA agreement governs the project; hence all the identifying information is anonymized.

Table 2.9: Characteristics of the Brightsquid projects active as of March 2020 and included in our dataset.

| Project ID | # Commits | # Files | # Bugfixes | # LOC | #of Developers |
|------------|-----------|---------|------------|---------|----------------|
| Proj1 | 69 | 8 | 11 | 981 | 5 |
| Proj2 | 178 | 84 | 210 | 4,509 | 2 |
| Proj3 | 213 | 67 | 74 | 21,741 | 5 |
| Proj4 | 12 | 6 | 8 | 123,873 | 9 |
| Proj5 | 422 | 54 | 105 | 30,161 | 5 |
| Proj6 | 80 | 11 | 18 | 1,331 | 7 |
| Proj7 | 555 | 42 | 53 | 45,528 | 6 |
| Proj8 | 1085 | 292 | 491 | 9,685 | 5 |
| Proj9 | 1350 | 339 | 740 | 50,918 | 9 |

After conducting a literature review, we identified the dependent, independent, and control variables from three studies. In particular, Bird et al.[56] examined ownership at the component level, defining components as a "unit of development that has some core functionality. [...] In Windows, a component is a compiled binary." After discussing with our industry partner, we considered the "source code file" as the component to replicate the analysis of ownership at the file level for answering RQ_{S71} and RQ_{S72} . We collected four main metrics for file ownership from nine projects at Brightsquid, following Bird et al.'s[56] methodology, as detailed in Table 2.7. We then retrieved the number of failures/bugs from the Brightsquid issue management repository on JIRA. By using integration tools between JIRA and GitHub, we traced issues back to commits and code, allowing us to gather bugs linked to each source code file. Interestingly, we found that the majority of files have no bugs assigned. Additionally, we gathered three code metrics – SIZE (measured by the number of lines of code in a file), COMPLEXITY (using cyclomatic complexity), and CHURN (as the lines of code changed in each file) – from the source code repository to support our analysis.

To replicate the study by Greiler et al. [68], we analyzed the same nine Brightsquid

projects at both the file and directory levels of source code. We measured individual file ownership using four metrics, which are detailed in Table 2.7. To investigate directory-level ownership, we aggregated the number of distinct bugs associated with all source files and used this information to infer the number of directory-level issues. However, unlike in the original study, we were unable to identify any file that had been edited only once. We found that all files in the selected Brightsquad projects had been either edited more than once or not at all since their creation. However, unlike the original study that included "organization ownership" metrics, we could not obtain equivalent data from Brightsquad. As a result, our analysis was restricted to individual ownership metrics.

We also collected one additional data point, "Touches," which measures the number of files touched by a developer, as introduced by Foucault et al. [75]. Overall, we gathered three unique dependent, twenty independent, and three controlled variables from the three studies we chose. These variables are listed in Table 2.7.

2.5.3 Empirical Results and Ownership Status Quo at Brightsquad

Figure 2.8 summarizes the ownership metrics in Brightsquad based on Greiler et al.'s [68] measurements. We analyzed ownership statistics for our selected projects using a 50% threshold to identify major and minor contributors. Figure 2.8 - (d) shows that at the file level, the majority of contributors are minor contributors. In most projects,

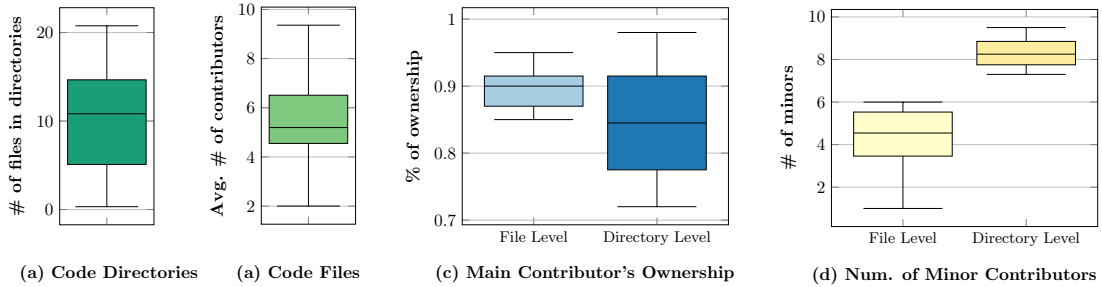


Figure 2.8: Descriptive statistics of ownership metrics of Brightsquad projects. The metrics calculated based on Greiler et al. [68] work.

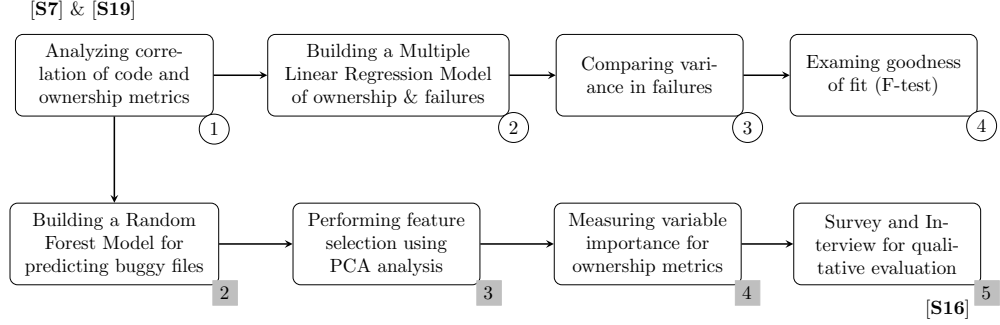


Figure 2.9: Main steps taken in the original studies of Bird et al. [6][S7], Greiler et al. [68][S16], and Foucault et al. [74][S19] to analyze ownership

only one contributor is major, with all others minor. However, we also found that the main contributor accounts for over 85% of the commits.

Figure 2.9 outlines the main steps of the methodology used in the three papers. We discuss the results of these steps as they relate to Brightsquad. At the directory level, we observed that the average number of files per directory ranges from a minimum of three to a maximum of 21 for Proj1 and Proj9, respectively. The majority of contributors are minor contributors, with over 75% of them being classified as such. The average percentage of ownership at the directory level is consistently above 72%, indicating that the main contributor made the majority of the commits to a file or directory and suggesting a strong sense of ownership at both levels.

Step 1 - Correlation of Code and Ownership Metrics: We followed the protocol of the three studies we considered and computed Spearman rank correlations between code ownership metrics and the NUMBER OF BUGS for each of our selected projects. This was designed to explore the relationship between ownership metrics and software quality. We investigated correlations between the NUMBER OF BUGS and all metrics, both at the file and directory level. We performed a Spearman correlation analysis of the NUMBER OF BUGS with four ownership metrics: OWNERSHIP, TOTAL, MAJOR, and MINOR, as well as three code metrics: CODE SIZE, CHURN, and COMPLEXITY, for nine projects of Brightsquad shown in Table 2.10. Note that we aggregated variables that were called differently but had exactly the same definition, such as LOC, NUMDEVS,

Table 2.10: Spearman correlation between metrics and the number of bugs on file level granularity. Bold and underlined values are absolute correlation coefficients above 0.50 and 0.75, respectively

| Metric | Proj1 | Proj2 | Proj3 | Proj4 | Proj5 | Proj6 | Proj7 | Proj8 | Proj9 | Avg. |
|------------------------|--------------|-------------|--------------|---------------------|--------------------|--------------|-------------|--------------------|--------------|--------------|
| OWNER SHIP (MVO) | -0.61 | -0.17 | -0.72 | <u>-0.89</u> | -0.75 | -0.59 | -0.66 | -0.66 | -0.59 | -0.63 |
| MAJOR | 0.67 | 0.22 | 0.73 | <u>0.93</u> | <u>0.81</u> | 0.62 | 0.71 | <u>0.81</u> | 0.61 | 0.68 |
| MINORS ($< 5\%$) | -0.13 | 0.24 | 0.03 | 0.08 | 0.27 | 0.36 | 0.26 | 0.28 | 0.19 | 0.17 |
| MINORS ($< 50\%$) | -0.39 | 0.60 | 0.21 | 0.85 | 0.54 | 0.88 | 0.53 | 0.61 | 0.41 | 0.47 |
| MINIMAL | -0.21 | 0.5 | 0.2 | 0.82 | 0.45 | 0.82 | 0.52 | 0.58 | 0.41 | 0.45 |
| TOTAL (NUMDEV) | 0.67 | 0.27 | 0.73 | <u>0.93</u> | <u>0.81</u> | 0.66 | 0.73 | <u>0.81</u> | 0.61 | 0.69 |
| TOUCH | 0.23 | 0.42 | 0.26 | -0.05 | 0.33 | 0.42 | 0.34 | 0.44 | 0.14 | 0.28 |
| Code metric | | | | | | | | | | |
| CHURN | 0.26 | 0.29 | 0.24 | 0.74 | 0.32 | 0.43 | 0.23 | 0.48 | 0.54 | 0.39 |
| SIZE (LOC) | 0.41 | 0.52 | 0.19 | -0.22 | 0.4 | -0.01 | 0.28 | 0.7 | 0.4 | 0.30 |
| COMPLE XITY | 0.69 | 0.18 | 0.04 | 0.53 | 0.29 | 0.22 | 0.26 | 0.21 | 0.44 | 0.32 |

and MVO, which are the same as SIZE, TOTAL, and OWNERSHIP, as defined by Bird et al. [6].

File Level Analysis: Table 2.10 shows the correlations at the file level. The results showed that except for one of the nine projects (Proj2), there is a significant correlation between the NUMBER OF BUGS and OWNERSHIP, MAJOR, and TOTAL metrics. As summarized in Table 2.10, for all the Brightsquid projects, OWNERSHIP is negatively correlated with the NUMBER OF BUGS, indicating that stronger file ownership correlates with fewer bugs. In other words, the more the code is shared among multiple developers (the lower the OWNERSHIP), the higher the likelihood that the application will contain bugs. Similarly, we found a rather strong positive correlation between the NUMBER OF BUGS and both TOTAL and MAJOR metrics. This indicates that

Table 2.11: Spearman correlation between metrics and bug numbers on directory level. Bold values are absolute correlation coefficients above 0.50.

| Metrics | Proj1 | Proj2 | Proj3 | Proj4 | Proj5 | Proj6 | Proj7 | Proj8 | Proj9 | Avg. |
|-----------------|--------------|--------------|--------------|--------|---------------|--------------|--------|--------|--------|-------|
| AVGOWNERSHIP | -0.310 | -0.008 | -0.137 | -0.04 | -0.185 | 0.630 | -0.043 | 0.106 | 0.136 | 0.016 |
| OWNERSHIPDIR | 0.188 | -0.328 | -0.127 | -0.047 | -0.069 | 0.579 | -0.213 | 0.075 | 0.137 | 0.022 |
| MINOWNERSDIR | 0.145 | -0.328 | -0.132 | -0.042 | 0.0255 | 0.463 | -0.109 | 0.082 | 0.147 | 0.028 |
| PCMINORS | 0.218 | -0.352 | -0.121 | -0.048 | -0.080 | 0.802 | -0.157 | 0.084 | 0.143 | 0.054 |
| PCMINIMALS | 0.306 | -0.352 | -0.125 | -0.047 | -0.130 | 0.617 | -0.226 | 0.077 | 0.136 | 0.028 |
| PCMAJORS | 0.218 | -0.352 | -0.134 | -0.042 | -0.047 | 0.802 | -0.142 | 0.022 | 0.150 | 0.052 |
| AVGMINORS | 0.262 | 0.034 | -0.135 | -0.047 | -0.163 | 0.630 | -0.084 | 0.108 | 0.138 | 0.082 |
| MINOWNERSDFILE | 0.599 | 0.083 | -0.66 | -0.049 | 0.062 | 0.011 | 0.063 | 0.242 | 0.135 | 0.054 |
| AVGMINIMALS | 0.320 | 0.034 | -0.136 | -0.04 | -0.136 | 0.630 | -0.136 | 0.107 | 0.138 | 0.087 |
| WEAKOWNERS | 0.623 | 0.344 | 0.102 | -0.024 | -0.051 | -0.037 | -0.088 | -0.132 | -0.201 | 0.059 |
| AVGCONTRIBUTORS | 0.66 | 0.506 | 0.16 | -0.28 | -0.571 | 0.626 | -0.08 | 0.281 | 0.255 | 0.173 |

a higher number of TOTAL and MAJOR owners correlate with a higher NUMBER OF BUGS.

When it comes to the MINOR metric, we consistently observed a weak correlation for all the projects. Even when changing the threshold of MINORS from 5% to 20% (MINIMALS as named by Foucault et al.[74]) and then to 50% (as suggested by Greiler et al.[68]), MINORS still does not show a consistently strong and significant correlation. Similarly, we only observed a sparse correlation between those and the NUMBER OF BUGS and code metrics in Brightsquid projects (see Table 2.10).

Directory Level Analysis: Table 2.11 contains Spearman correlation value for metrics on directory level. In the case of Brightsquid we only observed weak correlation between the ownership metrics for directory level and the NUMBER OF BUGS across

almost all projects of Brightsquad. Only, Proj6 presents a significant correlation value for the directory level. Also, the correlation result is not consistent for all projects. Within five projects the NUMBER OF BUGS decreases with the increasing percentage of MINOR, MINIMAL, and MAJOR contributors within a directory. For the other four projects of BRIGHSQUID it is the opposite; meaning that NUMBER OF BUGS increases with the increasing percentage of those metrics. While the results are indecisive at the directory level, we observed that the correlation between the THE NUMBER OF BUGS and ownership metrics is more significant on the file level.

In Brightsquad, stronger file ownership associates with fewer NUMBER OF BUGS at the file level. Further, the consistently strong correlation between the number of TOTAL and MAJOR owners correlate with higher NUMBER OF BUGS.

Step 2 - Multiple Linear Regression Model of Ownership and Failure: The correlation between the ownership and the code metrics motivated Bird et al. [6] to build regression models. It is essential to analyze if the increasing NUMBER OF BUGS in the projects is attributable to more MINOR contributors or to other measures such as SIZE, CHURN, and COMPLEXITY that are also known to be related to faults. Similarly, we used multiple linear regression to examine the relationship of ownership metrics while controlling source code attributes (here SIZE, CHURN, and COMPLEXITY). We built five statistical models for every project to examine how large or small the effect of each metric is on the NUMBER OF BUGS.

Step 3, 4 - Comparing Variance in Failure (\mathbb{R}^2) and Goodness of fit: After constructing the five models, we evaluated and compared their variance in failure (\mathbb{R}^2). We began with the BASE model consisting of only the three code metrics: SIZE, CHURN, and COMPLEXITY. We subsequently added each ownership metric one by one to determine which variables influenced the increase or decrease of bugs, following the approach of Bird et al. [6].

We summarized the results of this analysis based on our regression model for Brightsquad in Table 2.12. In this table, the asterisk (*) indicates cases where we iden-

Table 2.12: Variance in failures for the base model (Code metrics) which includes standard metrics of COMPLEXITY, SIZE, and CHURN, as well as the models with MINOR, MAJOR and OWNERSHIP added

| Project | Base | Base+total | Base+minor | Base+minor+major | Base+minor+major+ownership |
|-------------|--------|---------------|---------------|------------------|----------------------------|
| Proj1 | 38% | 38%(+0%) | 38.7%(+0.7%) | 38.9%(+0.2%) | 40.4%*(+1.5%) |
| Proj2 | 51% | 91%*(+40%) | 91%*(+40%) | 91%(+0%) | 91%(+0%) |
| Proj3 | 84.3% | 85.0%(+0.7%) | 85%(+0%) | 85.8%(+0.8%) | 85.8%(+0%) |
| Proj4 | 92% | 93.3%*(+1.3%) | 93.1%*(+1.1%) | 93.4%(+0.3%) | 97.6%*(+4.2%) |
| Proj5 | 56% | 56.04%(+0.4%) | 56.06%(+0.6%) | 57.4%(+0.8%) | 59.7%*(+2.3%) |
| Proj6 | 39.7% | 44.8%*(+5.1%) | 48.9%*(+9.2%) | 52.9%*(+4%) | 70%*(+18.1%) |
| Proj7 | 84% | 85.3%*(+1.3%) | 85.9%*(+1.9%) | 88.3%*(+2.4%) | 89.8%*(+1.5%) |
| Proj8 | 35.3% | 50%*(+14.7%) | 51.2%(+1.2%) | 51.2%(+0%) | 51.2%(+0%) |
| Proj9 | 42.5% | 47.7%*(+5.2%) | 48.1%*(+5.6%) | 49.7%*(+1.6%) | 49.9%(+0.2%) |
| Avg. | 58.09% | 65.68% | 66.44% | 69.62% | 69.48% |

tified that adding a variable significantly improved the model through the goodness-of-fit F-test [6]. Among the five models, the base model comprises only three code metrics SIZE, CHURN, COMPLEXITY. We observed that the BASE model and the three code metrics primarily explained variance for Proj4 and Proj7, with percentages of 92.0% and 84.0%, respectively. By adding TOTAL and then MINOR as predictors, we observed that both variables significantly increased the proportion of variance for all projects except Proj5. On average, adding the MINOR variable increased the variance in failure by 6.7%, while adding the TOTAL variable increased this number by 7.63%.

We added MAJOR and OWNERSHIP variables as predictors for building the fourth (BASE + MINOR + MAJOR) and the fifth model (BASE + MINOR + MAJOR + OWNERSHIP). However, we found that these two metrics had less of an effect compared to when we added TOTAL and MINOR metrics, especially for the Proj2 and Proj9 projects. Therefore, we can conclude that when controlling for code metrics, the number of TOTAL and MINOR contributors have a strong relationship with the *number of bugs*. Furthermore, MINOR contributors have a greater impact on increasing

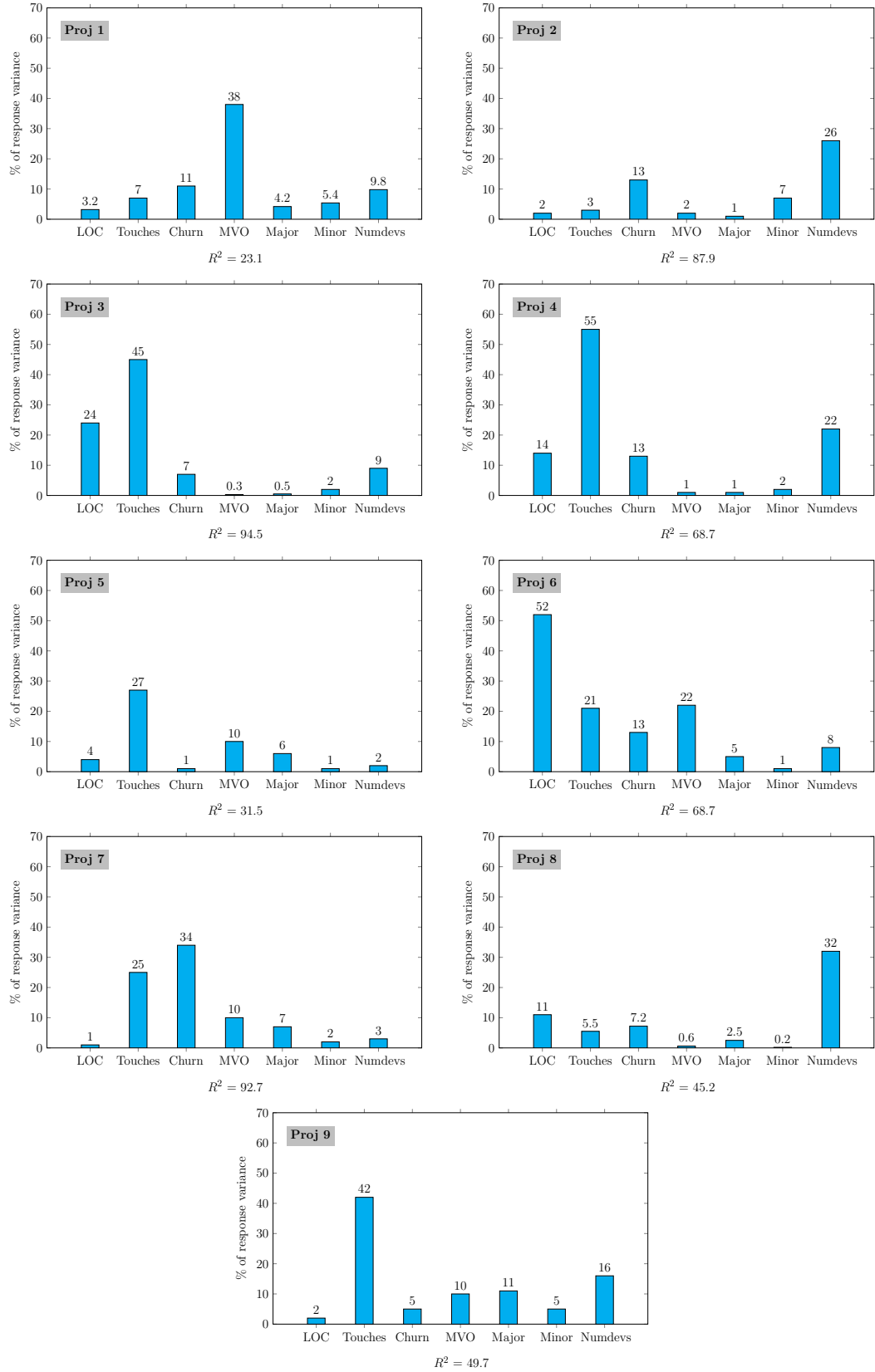


Figure 2.10: Relative importance of metrics in regression models using the NUMBER OF BUGS as the dependent variable.

the *number of bugs* compared to the higher level of OWNERSHIP. Our results also show a positive correlation between the TOUCHES metric and the NUMBER OF BUGS, except for Proj4. However, this correlation is relatively weak, with an average of 0.21.

When comparing the \mathbb{R}^2 values between LOC and CHURN and MVO, MAJOR, and MINOR, we observed that code metrics contribute relatively more to the total \mathbb{R}^2 compared to the other metrics. Figure 2.10 illustrates and compares these results.

*In **Brighsquad**, the number of TOTAL and MINOR contributors has a strong relationship with code quality when controlling the source code attributes. However, we did not observe strong prediction power for ownership metrics compared to the code metrics.*

Table 2.13: Details on Precision, Recall, and F-measure for predicting defective source files and directories.

| Directory level | | | | File level | | |
|-----------------|-----------|--------|----------|------------|--------|----------|
| Project | Precision | Recall | F1 score | Precision | Recall | F1 score |
| Proj1 | 0.70 | 0.62 | 0.65 | 0.36 | 0.34 | 0.35 |
| Proj2 | 0.92 | 0.89 | 0.90 | 0.65 | 0.61 | 0.63 |
| Proj3 | 0.94 | 0.85 | 0.89 | 0.95 | 0.86 | 0.90 |
| Proj4 | 0.62 | 0.60 | 0.61 | 0.60 | 0.60 | 0.60 |
| Proj5 | 0.95 | 0.92 | 0.93 | 0.92 | 0.85 | 0.88 |
| Proj6 | 0.93 | 0.92 | 0.92 | 0.95 | 0.93 | 0.94 |
| Proj7 | 0.95 | 0.95 | 0.95 | 0.89 | 0.88 | 0.88 |
| Proj8 | 0.90 | 0.88 | 0.89 | 0.93 | 0.91 | 0.92 |
| Proj9 | 0.86 | 0.81 | 0.83 | 0.87 | 0.83 | 0.85 |
| Avg. | 0.86 | 0.82 | 0.84 | 0.79 | 0.76 | 0.77 |

Step 2,3 - Random Forrest Model for Predicting Buggy Files and Directo-

ries: We built a Random Forest machine learning classifier to predict defective source files and directories. Following Greiler et al. [68], we divided our dataset into *files* and *directories* to investigate both granularities. We used two-thirds of one dataset for training and one-third for testing, along with 10-fold cross-validation. We also employed PCA (Step 3 - Figure 2.9) for feature selection and dimension reduction. Finally, we evaluated the classifier’s performance in terms of precision, recall, and F1

score.

Table 2.13 summarizes the Random Forest classifier’s performance in predicting defective sources at both the file and directory levels. At the directory level, we achieved an average F1 score of 0.84 (ranging from 0.61 to 0.93). At the file level, we obtained an average F1 score of 0.77 (ranging from 0.35 to 0.94). Our model performed relatively worse in terms of precision and recall at the file level than directory level. This classification result indicates that ownership can be a significant factor in code quality

The Random Forest model applied to Brightsquid demonstrated the potential for predictive modeling of code quality based on ownership values at both file and directory levels. However, the model performed poorly in predicting quality based on file ownership metrics for one of the nine projects.

Step 4 - Variable Importance in Ownership: Further, we conducted a metric importance analysis for both the *file level* and *directory level* to assess the predictive power of the various metrics. To perform this analysis, we utilized the Random Forest model implemented in `scikit-learn`, specifically the `Random Forest Regressor` model. We present the metric importance scores for the *file level* in Table 2.14 and for the *directory level* in Table 2.15.

On the file level, Table 2.14 shows that the metric with the highest importance score for predicting defective source files is the number of `CONTRIBUTORS`, except for Proj7 and Proj9. On the other hand, the least important metric for predicting defective source files varies depending on the project. For example, in Proj2, Proj3, and Proj8, `OWNERSHIP` is the least important metric, while in Proj1, Proj5, Proj7, and Proj9, it’s the `MINIMALS` metric.

Looking at the attribute importance on the *directory level* (see Table 2.15), `MINOWNEDFILE` and `AVGCONTRIBUTORS` have the highest importance scores for predicting defective source directories across all projects. However, the importance of other metrics is inconsistent across different projects. For instance, in Proj3 and Proj8,

all metrics except MINOWNEDFILE and AVGCONTRIBUTORS have poor importance scores.

The number of contributors to a file and the lowest ownership value among all the files in a directory show the highest predictive power among all at Brightsquad.

Table 2.14: Metric importance for prediction model based on ownership metrics classifying detective source files

| Metrics | Proj1 | Proj2 | Proj3 | Proj4 | Proj5 | Proj6 | Proj7 | Proj8 | Proj9 | Avg. |
|--------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| OWNERSHIP | 0.032 | 0.023 | 0.003 | 0.008 | 0.101 | 0.22 | 0.102 | 0.004 | 0.104 | 0.121 |
| MINORS | 0.109 | 0.078 | 0.007 | 0.007 | 0.061 | 0.112 | 0.078 | 0.008 | 0.116 | 0.064 |
| MINIMALS | 0.013 | 0.112 | 0.005 | 0.026 | 0.014 | 0.112 | 0.023 | 0.012 | 0.051 | 0.041 |
| CONTRIBUTORS | 0.121 | 0.268 | 0.095 | 0.221 | 0.022 | 0.085 | 0.034 | 0.087 | 0.165 | 0.122 |

Table 2.15: Metric importance for prediction model based on ownership metrics classifying source directories

| Metrics | Proj1 | Proj2 | Proj3 | Proj4 | Proj5 | Proj6 | Proj7 | Proj8 | Proj9 | Avg. |
|----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| AVGCONTRIBUTOR | 0.14 | 0.161 | 0.145 | 0.33 | 0.162 | 0.27 | 0.101 | 0.142 | 0.131 | 0.176 |
| PCMAJORS | 0.109 | 0.077 | 0.007 | 0.007 | 0.061 | 0.112 | 0.078 | 0.009 | 0.116 | 0.064 |
| AVGMINORS | 0.017 | 0.023 | 0.006 | 0.021 | 0.095 | 0.01 | 0.108 | 0.004 | 0.032 | 0.035 |
| WEAKOWNEDS | 0.148 | 0.099 | 0.006 | 0.020 | 0.021 | 0.046 | 0.086 | 0.004 | 0.059 | 0.054 |
| PCMINIMALS | 0.032 | 0.023 | 0.002 | 0.009 | 0.101 | 0.022 | 0.102 | 0.004 | 0.014 | 0.034 |
| PCMINORS | 0.013 | 0.112 | 0.005 | 0.027 | 0.01 | 0.112 | 0.023 | 0.011 | 0.051 | 0.040 |
| AVGOWNERSHIP | 0.10 | 0.028 | 0.009 | 0.005 | 0.123 | 0.124 | 0.009 | 0.002 | 0.088 | 0.054 |
| MINOWNERDIR | 0.07 | 0.164 | 0.008 | 0.004 | 0.08 | 0.115 | 0.111 | 0.003 | 0.008 | 0.063 |
| OWNERSHIPDIR | 0.10 | 0.09 | 0.006 | 0.006 | 0.018 | 0.104 | 0.023 | 0.001 | 0.068 | 0.046 |
| MINOWNEDFILE | 0.211 | 0.184 | 0.195 | 0.176 | 0.201 | 0.197 | 0.22 | 0.205 | 0.164 | 0.195 |
| AVGMINIMALS | 0.107 | 0.018 | 0.005 | 0.001 | 0.102 | 0.085 | 0.178 | 0.011 | 0.015 | 0.058 |

2.5.4 Comparing results between Brightsquad and the reported state of practice

Greiler et al. [68] replicated the study of Bird et al. [6] in `Microsoft` but expanded the analysis by introducing 12 new metrics for directory-level code ownership. They discovered a correlation between these metrics and the number of bugs in four `Microsoft`

products. Foucault et al. [74] built on this research with slightly different metrics and evaluated their model in open-source projects. After consulting with our partner company, Brightsquad, we selected these three studies as the most representative examples of code ownership research in practice. To evaluate code ownership in Brightsquad, we replicated these three studies and compared their findings with those from our partner company. Although we tested the hypotheses of these papers in Brightsquad, we adapted our evaluation to account for the specific context of our partner company. For example, we excluded metrics such as organizational and team ownership values that were not applicable in Brightsquad.

In their first hypothesis, all three selected studies tested whether components with many MINOR contributors have more bugs compared to others. In our analysis of the Brightsquad projects, we found a strong correlation between the NUMBER OF BUGS and the MAJOR, TOTAL, and OWNERSHIP metrics in all projects except Proj2. However, the correlation between the NUMBER OF BUGS and MINOR contributors was very weak for all nine projects. This is a completely different result from the original study, which suggested that software components with many MINOR contributors will have fewer failures. Even when changing the thresholds from 5% to 20% and 50%, the average correlation across the projects remained weak (i.e., ≤ 0.5), and the strength and direction of these correlations were inconclusive.

Comparison 1: In the case of Brightsquad, we observed that there was no significant correlation (Avg. = 0.17) between the NUMBER OF BUGS in software projects and MINOR contributors who had made contributions of less than or equal to 5%. This finding is contrary to the results reported in previous studies, which suggested a relationship between the number of bugs and MINOR contributors.

All of the studies we reviewed indicated a weak correlation between MAJOR contributors and the number of pre-and post-release failures. However, in our analysis of Brightsquad projects, we found a strong correlation between MAJOR contributors

and the NUMBER OF BUGS (Avg. = 0.677). Overall, previous studies tended to find a stronger correlation between code ownership metrics and the NUMBER OF BUGS. Similarly, in our analysis, we observed weaker correlations between code metrics and the NUMBER OF BUGS in Brightsquid projects compared to the studies we reviewed. For example, Bird et al. reported a relatively strong relationship between classic code metrics and the number of failures, but we only observed a sparse correlation between code metrics and the NUMBER OF BUGS in Brightsquid projects.

The product manager of Brightsquid stated that with his above 15 years of experience in startup companies having MINOR owners is unavoidable but also is essential:

“In order to support resiliency, smaller companies in emerging markets require minor ownership. i.e. all have an equal understanding of the full system.”

Comparison 2: In Brightsquid, we observed a correlation between the number of bugs and the number of TOTAL and MAJOR owners, independent of code metrics.

Foucault et al. [74] reported a significant correlation between the number of developers who modified a file (number of TOUCHES) and the number of bugs. However, in our analysis of the Brightsquid projects, we found no significant correlation between the number of file modifications and the number of bugs (Avg. = 0.28). Previous studies have tested the hypothesis that software components with a high level of OWNERSHIP have fewer failures compared to others. These studies showed a strong correlation between ownership metrics (MINOR, MAJOR, TOTAL, OWNERSHIP) and the number of bugs or failures. Consistent with these findings, we observed a negative correlation with an average of -0.63 between the OWNERSHIP metric and the number of failures in Brightsquid. We also found a strong correlation between TOTAL and the number of bugs, which is similar to the results of previous studies.

Greiler et al. [68] proposed 12 new ownership metrics at the directory level and reported a strong correlation between the number of bugs and the percentage of edits made by MINOR, MINIMAL, or MAJOR contributors (PCMINOR, PCMINIMAL,

PCMAJORS, MINOWNERDIR). They found that an increase in MINIMAL and MINOR contributors in a directory were associated with higher bug counts. They also reported that the number of bugs decreases if there are more MAJOR contributors among the contributors of a directory.

While having more MAJOR contributors in a directory was associated with lower bug counts. In contrast, our analysis of **Brighsquid** revealed stronger correlations between ownership metrics at the *file level* than those defined at the *directory level*. Specifically, we could not observe any significant correlation between any of the *directory level* metrics and the number of bugs in Brighsquid. Therefore, we cannot draw any conclusive results about the relationship between the number of bugs and the presence of MINOR, MINIMAL, or MAJOR contributors in a directory.

Comparison 3: Unlike the former studies in **Brighsquid**, we could not observe correlation between directory-level ownership metrics and the number of bugs.

Bird et al. [6] also hypothesized that the removal of MINOR contribution metrics decreases defect prediction model performance. They build a regression model and reported a statically significant impact of code metrics on the number of failures. They reported a clear trend of having a statistically significant relationship between ownership metrics and the number of failures in **Microsoft Windows**. In Brighsquid, our analysis showed a strong correlation between the number of TOTAL and MINOR contributors and code quality, even when controlling for source code attributes. However, we did not observe significant predictive power for ownership metrics compared to code metrics. Greiler et al. [68] took this a step further and evaluated whether component bugginess can be predicted using ownership and code metrics, using a Random Forest model. They reported an average F-score of 0.67, with recall performing poorly compared to precision. In our file-level analysis, we achieved an F-score of 0.77 on average and similarly observed poor recall compared to precision in our model.

Comparison 4: In Brightsquid, the number of developers is the most important factor for classifying defective source files. On the directory level, the lowest ownership value among all the files in a directory has the best power in predicting bugs.

In their studies, Foucault et al.[74] and Greiler et al.[68] assessed the significance of metrics for categorizing faulty source files and directories. Upon comparing our findings with theirs, we observed that the metric importance score for Brightsquid was very low. Nevertheless, similar to their studies, we discovered that the most important metric for classifying defective source files was CONTRIBUTORS. On the directory level, MINOWNERDIR exhibited the highest importance in *Microsoft* products, whereas in Brightsquid, MINOWNEDFILE scored the highest. While Foucault et al. [75] noted that LOC, TOUCHES, and NUMDEVS performed well in predicting the number of bugs in source files within open-source projects, our analysis of Brightsquid indicated that TOUCHES and NUMDEVS were the most effective metrics for predicting bugs in terms of relative importance.

2.6 Discussion and Implication of Findings at Brightsquid

We conducted a Systematic Literature Review (SLR) on ownership in software engineering, which revealed the existence of nine ownership artifacts. The most frequently discussed artifact in the literature was code ownership. We present and discuss our findings in this section, highlighting their implications. We in particular highlight three main implications of this literature review.

- The community and the active researchers in the field of ownership, from unifying the language and terminology used for modeling. While the terminology introduced by Bird et al. [6] has been widely used, there are numerous variables with identical meanings but different names. Appendix II provides details on the naming and definitions used in each paper. We further synthesized this data

using the most popular terminology, as shown in Table 2.4.

- Authorship refers to the process of determining who wrote a piece of software based on its source code. Traditionally, ownership of an artifact has been attributed to an individual (dedicated ownership), which is synonymous with authorship. However, with the advent of version control systems, shared ownership of artifacts is being represented either as a weighted value among team members or as a collective value among all team members. Code ownership and code authorship should be distinguished in terms of the method of measuring ownership.
- While the research community is strongly promoting replicability of empirical research the status is not very promising in the context of ownership as we evaluated in our systematic literature review. 84.8% of the ownership studies identified in our SLR are not replicable, while 67.1% of them are descriptive, and the majority present case study evaluations (see Figure 2.6).

Our SLR can serve as a starting point for future research in this context and help researchers select appropriate comparisons and terminology.

Brightsquid wanted to assess the status of code ownership in its repositories and compare it with other organizations. Our systematic literature review found two studies on proprietary software, both conducted at Microsoft [6], [68]. Bird et al. [6] examined the relationship between different ownership measures and software failures, introducing four ownership metrics ("Major", "Minor", "Total", and "ownership") and three code metrics ("size", "churn", and "complexity"). They then analyzed the correlation between these metrics and pre- and post-release failures. Our review identified four studies that evaluated code ownership, all of which followed the approach of Bird et al. [6]. In the majority of cases, our findings were not convergent with those reported in the literature.

However, when discussing our findings, we need to consider the context in which software engineering activities take place. Brightsquid is in essence focusing on new market/new solution segments as stated by their Chief Information Officer (CIO):

*“Our design and development methodologies must be much more adaptive and experimentation-driven. This is in stark contrast with the **Microsoft Windows** team, whose model is highly prescriptive because they’re optimizing an existing product for an existing market.”*

This is critical for understanding and improving the software development process [76]. Our findings echo the empirical evidence that shows yet another aspect of global and local defect prediction models. As put by Menzies et al. [77], simple global rules are inadequate for managing complex entities like defect prediction and effort estimation. Moreover, it is insufficient to describe the data by merely dividing it into local contexts. In the context of **Brightsquid** we identified that while concerned by the code quality, the company values developers accountability toward their costumers higher. Hence, when they receive and triage a costumer request [3] they assign that to one developer. The developer being assigned to the user request, is then responsible to make the decisions (either working with others or individually) to accommodate that request. This is the main reason for having multiple MINOR developers per file or directory. The company believes that understanding the customers and the context of their requests has a significant impact on a successful development:

- The strategy for assigning customer requests have changed the dynamic of ownership in **Brightsquid**. Focusing on user stories and requests rather than assigning the issues (or development tasks) to the developers results in significant number of MINOR developers working on a file/directory.
- Our results showed that the increase in the number of MAJOR contributors results in more number of bugs in **Brightsquid**. This is while the former studies

showed that the number of major contributors has a negative correlation with code bugginess. It is essential to consider the context of the software where Bird et al. [6] and Grailer et al. [68] performed analysis on **Microsoft** products including **Windows**. These products by their nature are less user driven in the core and mostly engineering design of an enterprise solution. Whereas in **Brighsquid** understanding the context of the user and their request is playing a pivotal role in a successful implementation of the changes or new features. In **Brighsquid** while the team structure is flat, MAJOR contributors has less familiarity and contact with individual customers. Brighsquid’s product manager indicative of feature/code change or evolution by developers:

When developing new solutions for new markets, we have to deal with both market risk (is our customer base interested in your feature?) and Technology risk (can we build the feature at an appropriate degree of quality?).
Consequently, MAJOR contributions mean the feature has been more volatile.

- Yet, our results show that the number of developers contributing to a file (CONTRIBUTORS) has an impact on the number of failures. While we could not identify the percentage of contribution as a conclusive factor **Brighsquid** should seek a balance between the expertise of the developers with maintaining particular parts of code (files and directories) and their familiarity with the user problems.

Our findings indicated comparable outcomes regarding the correlation between robust file ownership and a reduced number of bugs at the file level, along with the correlation between the count of principal owners and a higher number of bugs. However, unlike prior research, our observations did not reveal a substantial predictive capacity for code ownership metrics in forecasting code quality. Brighsquid’s product managers reflected on this as a contrast between the impact of process and personnel:

Greater developer ownership must be balanced against greater developer responsibility to acceptance criteria. I wonder if this suggests that "process" impacts quality more than "people"?

As recommended by Briand et al. [76], traditional approaches for conducting software engineering research which are focused on producing generalizable results, may not always be the most effective approach. Instead, the specific development contexts for **Brighsquid** are highly important. Hence, the previous studies might not be applicable to the specific software projects undertaken by **Brighsquid**, or the methodologies used in those studies might not have been appropriate for the context of the company's software projects. The difference in the perspective of task assignment in the team is the main difference that impacts the code ownership in **Brighsquid**. The difference between our observations and former studies suggests that other factors, such as the complexity of the software, the skills of the team members, or the quality of the testing processes, might have a more significant impact on the number of bugs in the software than the contributions of MINOR contributors. Moving forward we are planning to empirically evaluate the impact of development processes on the code ownership in **Brighsquid**.

As a follow up collaboration and based on the, we hope to further invest on test ownership in **Brighsquid**. The CIO of the company reflected on these results as;

Looking into the results I believe we should further invest on interface and unit test "contract" diligence as a greater predictor of software defects (at least in Brighsquid), in addition to the variables identified in the paper. Diligently defined, maintained and owned test cases/suites (by developers, not by separate testers) are predictive indicators of quality. With greater freedom (co-design/co-development) comes greater responsibilities (i.e. diligence to well-defined contracts). Further analyzing test case ownership, to me fits into the construct of the strong governance which perhaps needs to be in place to enable co-design/co-development. So the more

adaptive and evolutionary your solution is, the more important the maintenance of these test cases becomes.

2.7 Threats to Validity

Threat to validity refers to the limitations or potential biases in research that can affect the validity and reliability of the results. We discuss the threats to the validity of our study.

Are we measuring the right thing? Construct validity in systematic literature reviews (SLR) refers to the degree to which the review’s operational definitions of the key concepts or constructs being studied are accurate and complete. When performing a systematic literature review, the use of correct search keywords plays a critical role in identifying relevant studies. We identified 17 relevant search strings, which is a relatively high number compared to SLRs in software engineering. The manual analysis of many papers and the potential for human errors is another potential construct validity issue, which we believe we minimized by having at least two authors independently perform every classification.

When it comes to mining Brightsquad data, we mostly followed the approaches of the three papers introduced and the measurements suggested by these studies. The majority of the measurements and variables have been tested by multiple studies, yet there were a few (such as 'Touches') that have been proposed by only one of the studies. Nevertheless, we independently evaluated the potential relationship between these variables and software ownership in Brightsquad.

Are we drawing the right conclusions about treatment and outcome relation? Conclusion validity in a systematic literature review refers to the extent to which the conclusions drawn from the review accurately represent the evidence found in the studies included. It is concerned with whether the conclusions are supported by the evidence and whether alternative explanations for the findings have been ruled

out. When performing the SLR, We formed our research questions to identify the who, when, how and what code ownership have been discussed and identified different modeling aspects by systematically retrieving and unifying the dependent, independent, and controlled variables. Yet, there might be aspects that we missed in this comparison. We also analyzed all the active projects of Brightsquad at the time of this study. There were cases in that projects show slight differences in their behavior (for example correlation between the variables). These can be attributed to several socio-technical aspects and staffing of the project (for instance human resource retention) which were not evaluated nor quantified in our study.

Can the results be generalized beyond the scope of this study? External validity in systematic literature review refers to the generalizability of the findings to the larger population outside of the studies included in the review. It assesses the extent to which the results can be applied to other settings, populations, or contexts. Our study was limited in scope as we only examined active projects from our partner company. This means that our findings may not be applicable to all the incoming projects or the projects done in other organizations, particularly open-source projects. Furthermore, the development process for different projects can vary, and the results of our study are only applicable to projects written in Java.

Can we be sure that the treatment indeed caused the outcome? For an SLR, Internal validity refers to the extent to which the review methodology was applied rigorously and the study results are accurate and reliable. We chose a comprehensive set of databases to search in our systematic literature review. However, there is a risk that there exist other publications that are not indexed by these engines. Similarly, despite the use of 17 different search strings, the success of this literature review highly depends on the correct choice of keywords.

When analyzing Brightsquad data, we adopted the hypothesis that there is one developer as an owner of a project. Yet, this was mostly adopted from open-source projects. In Brightsquad, we observe that most developers are minor contributors

across different projects. Secondly, **Brighsquid** does not maintain the organizational metrics and the structure of the teams makes these factors irrelevant to this study. Hence, we only measured individual-level ownership and not team-level ownership. The absence of organizational metrics leaves room for speculation and interpretation, which could affect the accuracy of the observation.

2.8 Conclusion

Researchers have proposed various models to establish ownership of software artifacts and their relationship with the performance metrics of developers and code quality. However, there is a limited number of studies available on proprietary software, as the majority of the models have been empirically evaluated on open-source repositories. Despite this, no systematic study is currently available to provide an overview and taxonomy of the existing models for code ownership in proprietary software. our Systematic Literature Review (SLR) on ownership in software engineering revealed the existence of nine ownership artifacts, with code ownership being the most frequently discussed. Our findings suggest that unifying the language and terminology used for ownership modeling can benefit the discipline, as numerous variables with identical meanings but different names have been identified. Additionally, we found that authorship and ownership should be distinguished, as ownership can either be a shared value among team members or a dedicated value assigned to an individual. However, we also found that the status of replicability in ownership studies is not very promising, with 84.8% of the studies identified in our SLR being non-replicable. Furthermore, the majority of ownership studies are descriptive and present case study evaluations.

We identified three relevant research papers on code ownership for our partner company, **Brighsquid**. All three studies followed the approach of Bird et al. [6] which were conducted at Microsoft. After replicating these studies and adjusting our evaluation to the context of **Brighsquid**, we found that unlike the state-of-the-art

studies on code ownership, there is no significant relation between the number of minor owners and the code ownership status at Brightsquad. However, consistent with the former studies, the number of contributors has a significant impact on the bugginess of the source code. Overall, our study can serve as a starting point for future research in this context and help researchers select appropriate comparisons and terminology. Given that only 3.79% of ownership studies are replicable through the provision of their code and data packages, there is a need for more rigorous research methods and standardization in this field to establish baselines

Future work in this area could focus on developing more standardized methods for studying code ownership in proprietary software, exploring the relationship between ownership and other performance metrics, and investigating how ownership models can be integrated into software development processes. Additionally, researchers could investigate the factors that contribute to the low replicability of ownership studies and develop guidelines for improving replicability in this area.

Chapter 3

Ownership in the Hands of Accountability at Brightsquid: A case study and a Developer survey

The COVID-19 pandemic has accelerated the adoption of digital health solutions. This has presented significant challenges for software development teams to swiftly adjust with the market need and the demand. To address these challenges, product management teams have had to adapt their approach to software development, reshaping their processes to meet the demands of the pandemic. Brightsquid implemented a new task assignment process aimed at enhancing developer accountability towards the customer. To assess the impact of this change on code ownership, we conducted code change analysis. Additionally, we surveyed 67 developers to investigate the relationship between accountability and ownership more broadly. The findings indicate that the revised assignment model not only increased the perceived sense of accountability within the production team but also improved code resilience against ownership changes. Moreover, the survey results revealed that a majority of the participating developers (67.5%) associated perceived accountability with artifact ownership.

3.1 Introduction

The emergence of the COVID-19 global pandemic has accelerated the transformation of digital health solutions, as the increasing demand for online appointments and remote healthcare services became more pressing due to social distancing guidelines. This sudden shift has resulted in significant changes in the digital health industry and has added new challenges for software development teams, particularly in the healthcare sector. Former studies showed that the COVID-19 pandemic has had a profound impact on software development teams [78]–[81]. In particular, the surge in demand for digital health solutions has created a new set of challenges for software teams, including shifting priorities, increased workload, and rapidly changing customer demands. In response to these challenges, product management teams have had to adapt and evolve their approach to software development. To overcome the challenges posed by the pandemic, software development teams have had to change the shape and format of their product management processes.

Brightsquid¹ is specialized in providing secure communication solutions for the healthcare industry. The company’s mission is to help healthcare providers communicate more effectively and securely, enabling them to improve patient outcomes and deliver better quality care. Brightsquid’s flagship product, the Secure-Mail platform, is HIPPA² compliant and offers secure messaging and file-sharing capabilities, enabling healthcare professionals to exchange protected health information in a safe and efficient manner. Brightsquid has been involved in several projects and has experienced an increasing demand for new or enhanced versions of their existing solutions during the COVID-19 pandemic. The company’s product management team has made changes to their issue assignment model to increase developers’ accountability toward end-user, responding to the increased market demand.

The company explains this change as the transition from task assignment to design

¹<https://Brightsquid.com/>

²HIPAA: Health Insurance Portability and Accountability Act of 1996

assignment. In this new approach, a user story is assigned to developers, and it is their responsibility to identify relevant tasks and expert team members. While the team perceived an increase in developers' accountability, we initiated the partnership to evaluate the impact of this change on code ownership and code quality, and to what extent it affects them. We started with an interview with the product manager and the Chief Technology Officer (CTO) of Brightsquad to identify the problem.

3.1.1 Developer Accountability in the Face of High Customer Demand

With the rise in demand during the COVID-19 pandemic, the company recognized the critical importance of enhancing development team accountability toward customers. The team leads and product managers have increasingly noticed that developers tend to limit their contributions to simply carrying out assigned tasks, as suggested by the leads. The company has identified the significance of fostering a culture of co-creation to promote enhanced product innovation and increase customer satisfaction. As a result, the team opted for a change that holds developers further accountable for the outcomes of an implementation task:

“We intended to build a self-organizing team, which resembles how jazz musicians organize themselves. We have a broad structure and a shared destination that everyone understands and should commits to. Despite not having been together (during COVID), we have embraced this approach of self-organization. With the intent to bring more knowledge, curiosity, diligence, code ownership, and code robustness to the development team Brightsquad gradually changed the task assignment process to be more flexible in the last six months.”

To achieve this vision, the team leads emphasized a shift from task assignment and ownership to user story assignment and design ownership. Figure 3.1 - (a) and Figure 3.1- (b) illustrate the process before and after the assignment shift. Before the shift

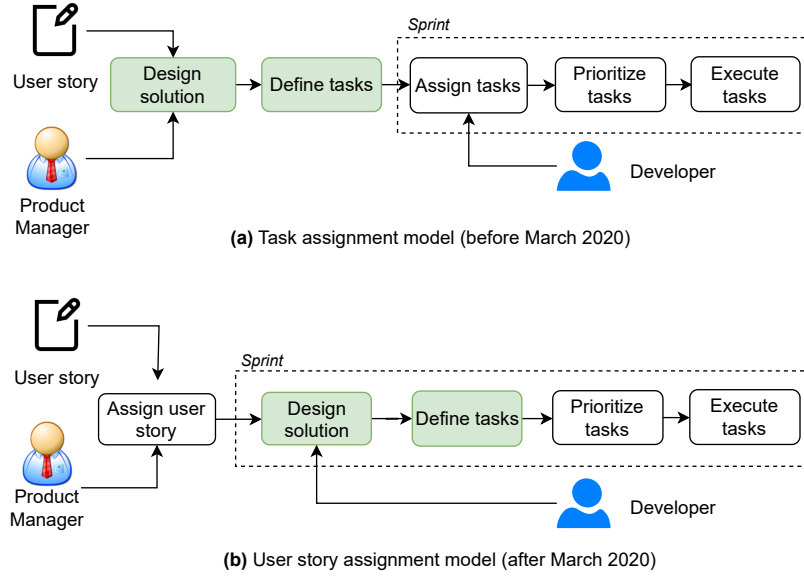


Figure 3.1: The process of (a) product manager details tasks for every user scenario and assign to a developer - Before March 2020 (b) product manager assign user story to a developer who is then responsible to design solution and define tasks.

in the task assignment process, the company followed a standard approach where the product manager identified the user story and collaborated with the users, market team, and customer management team to determine the specifics of the user story. The product manager then identified the tasks required to implement that user story. Subsequently, in a collaborative effort with the developers, the tasks were prioritized and assigned to developers in a sprint. The team has been practicing agile practices such as self-assignment following the agile practices [82].

However, after the change (Figure 3.1 - (b)) in the process, once the product manager identifies and details the user stories, they then identify the most suitable developer based on expertise and assign the story to them. From there, the developer takes on the responsibility of making decisions regarding design and implementing solutions. They identify the necessary tasks, enter them into the issue-tracking system, and prioritize them. Additionally, they collaborate with other team members to complete the assigned tasks. The product manager remains involved in the process to provide insights from customers, the customer management team, and the business perspective. In this scenario, the developer assigned to the user story also assumes accountability

for finalizing and reporting on the story’s progress. The company perceived a general value in this change, as put by Brightsquid’s CTO:

“It is the developers’ responsibility to get people together to talk about the design solutions or add subtasks under that user story based on their solution. (By this change) we observed the greater collaboration and participation. Developers were no longer just being order takers but also participating actively in figuring out what ought to be done.”

While the process aligns with the established transition of management in agile methodologies [83], the organization had only a primitive understanding and description of developers’ accountability and how to measure it. As framed by the product manager:

“Generally the idea was that whoever is assigned accountability for a particular story, if it succeeds, it reflects positively on them, but if it fails, it’s like, ‘Hey, you didn’t fulfill your responsibilities’.”

In this context and as a result of the semi-structured interview and its comparison with the literature, we formulate three research questions to evaluate the impact of this change on code ownership. To further gain a deeper understanding of the significance of accountability among developers. We aimed to explore a more precise definition of accountability in software teams and examine its relationship with code ownership among software developers.

3.1.2 Research Questions

Specifically, this paper is focused on answering the following research questions:

RQ1: How has the relationship between ownership and software quality metrics changed following the implementation of the new issue assignment model at Brightsquid?

While Brightsquid aimed to foster a sense of accountability among developers, the impact of this change on the quality of the product was unclear. Therefore, we are interested in comparing the code ownership status of Brightsquid’s project and the impact it has on the quality of the code before and after the task assignment change. We chose state-of-the-art studies and replicated them within BrightSsquid to measure the code ownership status in relation to code quality before and after the process change in March 2020.

RQ2: How did the revision of BrightSquid’s issue assignment process affect the performance of prediction models in identifying defective files and directories?

We aim to investigate the efficacy of ownership metrics in developing a classification model for predicting defective files and directories in Brightsquid projects. To assess the impact of the new assignment model implemented in March 2020, we compare the performance of these models before and after the process change, evaluating whether the new model improves the predictability of defects. Furthermore, we analyze the significance of ownership metrics for both files and directories to determine the extent to which different metrics can predict code business before and after the process change.

RQ3: How do developers perceive the relationship between accountability and ownership?

The relationship between accountability and ownership is a complex and multifaceted issue that is perceived differently by developers depending on a variety of factors. We surveyed 67 participants to explore their interpretation of accountability and ownership and the relation between these two as they perceive.

3.2 Brightsquid Data

Brightsquid has successfully completed 39 projects with the collaboration of 52 developers between 2012 and 2023, each project commencing at different times. The

company employs **GitHub** as a repository for storing its source code and leverages **JIRA**, an effective issue-tracking system, to efficiently manage its diverse range of issues. As of the time of writing this paper, Brightsquad had amassed an impressive count of 10,110 JIRA tickets across eight distinct categories: Story, Document, Task, Bug, New Feature, Epic, Technical Debt, and Improvement. These categories consist of 180 Epics, 1,243 Stories, 5,170 Bugs, 1,727 Improvements, 1,135 Tasks, 34 New Features, 548 Documentation, and 73 Technical Debts. To ensure a structured approach, the Story and Task issues are further classified into sub-stories, sub-tasks, and technical tasks, amplifying the organization’s ability to address complex problems. Meanwhile, Bug, Epic, and Improvement issues are divided into sub-stories and technical tasks, enabling a systematic resolution of these concerns. Notably, Technical Debt, New Feature, and Documentation issues do not possess any subdivisions, streamlining their management. Figure 3.2 provides an overview of this structure.

For our paper, we focused on four significant projects from Brightsquad that remained highly active between 2018 and 2022. These four projects contain a total of 1,515 bug fixes. We collected data for two distinct time frames: March 2018 to March 2020, prior to the implementation of the new accountability system, and March 2020 to March 2022, following the change. During our considered time frame March 2018 to March 2020, Proj4 has the most commits (1,350) and bugfixes (740) while Proj2

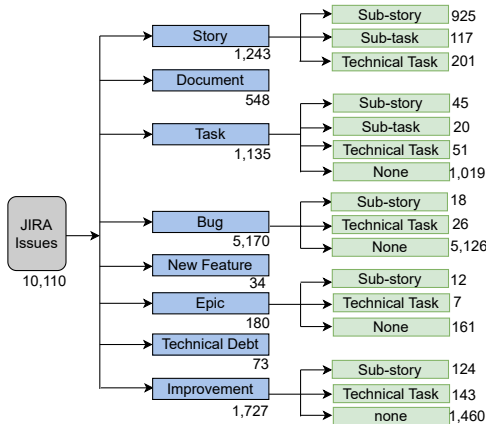


Figure 3.2: JIRA issue hierarchy of Brightsquad

Table 3.1: Characteristics of the studied Brightsquid projects.

| | March 2018 to March 2020 | | | | March 2020 to March 2022 | | | |
|-------|--------------------------|-------|---------------|------|--------------------------|-------|---------------|------|
| ID | Commits | Files | Changed files | Bugs | Commits | Files | Changed files | Bugs |
| Proj1 | 125 | 75 | 20 | 200 | 53 | 34 | 11 | 10 |
| Proj2 | 131 | 60 | 48 | 64 | 82 | 14 | 9 | 10 |
| Proj3 | 768 | 96 | 177 | 344 | 317 | 96 | 53 | 147 |
| Proj4 | 1,122 | 312 | 118 | 640 | 228 | 282 | 65 | 100 |

fixed the least bugs (74). On the other hand, Proj1 and Proj3 have 178 and 1,085 commits and 210 and 491 bugfixes respectively. For these four projects Table 3.1 shows the summary of these projects indicating the number of commits, total files, and files that have been changed within the time period, bug fixes, and number of developers for two distinct time periods.

3.3 Background

Software engineering has a significant body of research on code ownership [34], [52], [53], and Nordberg et al. [84]. For our research, we adopted the methodologies employed in two prominent studies that examined ownership status before and after March 2020. Bird et al. [6] conducted a comprehensive study at **Microsoft**, later replicated by Greiler et al. [68], which focused on ownership metrics and their impact on software quality. We selected these studies as they were conducted within a similar context to ours at Brightsquid, involving proprietary products.

Bird et al. [85] introduced four ownership metrics at the file level being Major, Minor, Ownership, and Total, which are further explained in Table 3.2. They investigated the relationship between ownership metrics and pre- and post-release bugs for **Windows Vista** and **Windows 7**. In their study on ownership, Bird et al. [6] demonstrated that software components with numerous Minor contributors tend to experience more failures compared to components with fewer contributors. Conversely,

Table 3.2: Definition of metrics inferred from literature and used in our study.

| Metric | Definition |
|-----------------|---|
| Major | # of developers who committed more than 5% to a file |
| Minors | # of developers who committed less than 5% to a file |
| Minimals | # of developers who made less than 20% commits |
| Modests | # of developers who made less than 50% commits |
| Total | # of developers who contributed to a file |
| Ownership | Proportion of commits for the highest contributor to a file |
| Avgownership | Average ownership values for all files in a directory |
| Ownershipdir | % of commits of the highest contributor among all files in a directory |
| Minownerdir | % of commits of the lowest contributor among all files in a directory |
| Avgcontributors | Average of distinct contributors among all files in a directory |
| Pcminors | % of contributors among all contributors with less than 50% commits for all directory files |
| Pcminimals | % of contributors among all contributors with less than 20% commits for all directory files |
| Pcmajors | % of contributors among all contributors with more than or 50% commits across all directory files |
| Avgminimal | Average minimals in a directory |
| Avgminors | Average minors in a directory |
| Minownedfile | The ownership value of the file with the lowest ownership value |
| Weakowneds | # of files in a directory that have an ownership value of less than 50% |

components with a high level of Ownership tend to have fewer failures than those with lower ownership levels. The researchers also found that contributors classified as Minor for one component often become Major contributors for other components that share dependency relationships. Furthermore, their study revealed that removing Minor contribution information from defect prediction techniques significantly reduces their performance. Greiler et al.[68] replicated the former study by Bird et al.[6] and expanded upon it by introducing 11 ownership metrics at the directory level. They developed a classification model to predict bugs and assessed the bug predictive power of the ownership metrics for both files and directories. Additionally, Greiler et al. adjusted the threshold for identifying Modest ownership, considering

ownership below 50% as the criterion. They also introduced a new concept called Minimal ownership, which referred to developers with less than 20% ownership. We provided a summary of all the ownership metrics in Table 3.2.

3.4 Empirical Methodology

The primary objective of our study is to compare the ownership status and metrics before and after Brightsquid transitioned from assigning tasks to assigning user stories to software developers. This shift in the process serves as a key focus of our analysis, allowing us to evaluate the impact on ownership and associated metrics.

3.4.1 The relationship between code ownership and code quality metrics (RQ1)

We start by calculating the Spearman rank correlations between code ownership metrics and the number of bugs across the four projects. Our objective is to investigate the relationship between ownership metrics and software quality. We examined these correlations at both the file and directory levels. We explored the association between the number of bugs and four ownership metrics: Ownership, Total, Major, and Minor. We constructed a multiple linear regression model to examine the relationship between ownership metrics and the number of bugs. This model enables us to determine whether the increase in the number of bugs within projects can be attributed to code ownership metrics (for example, higher involvement of minor contributors) or to code-related factors (for example code size, churn, and complexity), which are known to be linked to software faults. By utilizing multiple linear regression, we analyzed the association between ownership metrics and bugs while accounting for source code attributes, specifically size, churn, and complexity. For each project, we created five statistical models to assess the extent of the impact that each metric has on the number of bugs. Initially, we established the “Base” model, which incorporated the three code metrics size, churn, and complexity. Subsequently, we iteratively add each

ownership metric individually to identify the metrics that exerted an influence. We compare these models in terms of their variance in bugs (\mathbb{R}^2).

For both calculating the correlation and constructing the linear regression models, we divided the data into two segments before and after March 2020. This division was made to account for the change in the assignment model implemented by Brightsquid. We then compared the correlation and the variance in bugs measures for each project and across all the projects. Also, we investigated **RQ1** at both the file level and directory level. Bird et al. [6] conducted an analysis of **Microsoft Windows** products using file-level binaries. Following that, Greiler et al. [68] conducted a study that examined ownership in **Windows** products at the directory level. Their results showed an enhanced performance of defect prediction models when ownership was evaluated at the directory level.

3.4.2 Bug predictability performance (RQ2)

To investigate RQ2, we initially utilized a Random Forest classifier to predict defective source files and directories in Brightsquid. This choice of approach was influenced by the methodology employed by Greiler et al. [68] when analyzing ownership in **Microsoft**. The performance of our models was evaluated using 10-fold cross-validation. To assess the impact of the assignment process change on defect predictability, we constructed separate models for two distinct time periods: the two years prior to March 2020 and the two years following it. This division enables us to examine whether the predictability of defects improved (or not) after the assignment process change. Furthermore, we conducted training and testing of our models at both the file level and the directory level to provide a comprehensive analysis.

To conduct a more comprehensive analysis and evaluate the prediction power for each metric, we performed a metric importance analysis to evaluate the predictive capability of different metrics at the file and directory levels. In this evaluation, we specifically focused on the files and directories that underwent changes. The analysis

was carried out using the Random Forest Regressor model from the `scikit-learn` library in `Python`. We then compared the predictive power of these metrics before and after March 2020, when Brightsquad changed their assignment model. This analysis allows us to assess the impact of the assignment model change on the effectiveness of the metrics in predicting defects.

3.4.3 Developer’s perception of ownership and accountability in software teams (RQ3)

we discussed the results of **RQ1** and **RQ2** and their implications with the Brightsquad team. In order to gain a broader understanding of developers’ perceptions in general, beyond just Brightsquad, and to investigate the relationship between code ownership and accountability in software teams, we have performed survey research following the guidelines outlined by Pfleeger and Kitchenham[86]. Our survey research consists of four major parts. The first part aimed to collect participants’ demographics. The second part aimed to investigate the source and degree of a developer’s sense of ownership within a team then we aimed to understand the definition of accountability at work for software developers. Lastly, we investigated the relationship between ownership and accountability. We designed the survey together with the Brightsquad team to evaluate their hypothesis in a broader population.

In total, the survey consisted of 26 questions, including 24 closed-ended questions and two open-ended questions. We collected demographic information through four questions. The remaining questions aimed to gather participants’ opinions, experiences, and decisions using a five-point Likert scale, multiple-choice options, and text boxes. We obtained ethics approval for this survey from the York University board of ethics with Certificate # : e2023 – 088. The survey was anonymous, and no personal information was collected from participants. We used `Qualtrics` as the survey instrument. We used convenient sampling for attracting participants [86] and distribute the survey in our social networks. Overall, 128 times the link was clicked and

67 developers participate in our survey.

3.5 Results

In this section, we respond to each research question in sequence.

3.5.1 Relationship between code ownership and code quality metrics (RQ1)

We calculate the correlation between ownership metrics (see Table 3.2) and the number of bugs for each project at Brightsquad. We did this analysis for files and for directories. We did this separately for the time period of March 2018 to 2020 and the period of March 2020 to March 2022. We first present file-level results and follow the directory-level evaluations.

Table 3.3: Spearman correlation between metrics and bug numbers on file level.

| | March 2018 to March 2020 | | | | | March 2020 to March 2022 | | | | |
|--------------------------|--------------------------|--------------|--------------|--------------|-------------|--------------------------|--------------|--------------|--------------|-------------|
| Ownership Metrics | Proj1 | Proj2 | Proj3 | Proj4 | Avg. | Proj1 | Proj2 | Proj3 | Proj4 | Avg. |
| Ownership | -0.217 | -0.214 | -0.394 | -0.432 | -0.314 | -0.334 | -0.261 | -0.422 | -0.457 | -0.369 |
| Major (> 5%) | 0.236 | 0.259 | 0.43 | 0.548 | 0.368 | 0.349 | 0.266 | 0.433 | 0.474 | 0.380 |
| Minors (< 5%) | 0.236 | 0.213 | 0.393 | 0.429 | 0.317 | 0.334 | 0.261 | 0.422 | 0.465 | 0.370 |
| Modests (< 50%) | 0.236 | 0.214 | 0.40 | 0.43 | 0.320 | 0.296 | 0.235 | 0.192 | 0.102 | 0.206 |
| Minimals (< 20%) | 0.217 | 0.214 | 0.184 | 0.425 | 0.26 | 0.334 | 0.261 | 0.174 | 0.317 | 0.271 |
| Total (NumDevs) | 0.236 | 0.237 | 0.412 | 0.50 | 0.346 | 0.343 | 0.264 | 0.428 | 0.465 | 0.375 |
| Code Metrics | | | | | | | | | | |
| Churn | 0.211 | 0.233 | 0.416 | 0.539 | 0.350 | 0.434 | 0.265 | 0.428 | 0.477 | 0.401 |
| Size | 0.438 | 0.191 | 0.464 | 0.293 | 0.346 | 0.392 | 0.365 | 0.093 | 0.339 | 0.297 |
| Complexity | 0.217 | 0.214 | 0.184 | 0.425 | 0.26 | 0.334 | 0.261 | 0.174 | 0.317 | 0.271 |

At the file level, Table 3.3 presents a comparison of the Spearman correlations

between ownership metrics and the number of bugs at the file level before and after March 2020, when the assignment model changed in Brighsquid. In both time periods, we observed a negative correlation between the metric “ownership” and the number of bugs in the projects consistently across all projects. This indicates that higher levels of code ownership are associated with a decrease in the number of bugs. Furthermore, we observed a stronger negative correlation between bugs and ownership for all the projects after the assignment model changed (after March 2020). Additionally, we found that contributors classified as “Minor,” “Minimal,” and “Modest” all showed a positive correlation with the number of bugs in the code across all projects. However, none of the correlations are strong (all are < 0.5).

Furthermore, we discovered a positive correlation between the number of “Major” contributors and the number of bugs, which became stronger with the new assignment model. These findings align with previous studies conducted by Bird et al. [6] and Greiler et al. [68] in **Microsoft** projects. We also observed that all the correlations became stronger after implementing the new assignment model, except for the correlation between “Modest” contributors ($< 50\%$) and code size with the number of bugs, as reported in Table 3.3.

At the directory level, Table 3.4 presents a comparison of Spearman correlation values for metrics during the two time periods. Although the correlations generally become stronger after March 2020 and with the change in the ownership model, they all remain insignificant and close to zero. This is in contrast to the analysis conducted by Greiler et al. [68], which showed much stronger correlations between these metrics and the number of bugs in **Microsoft**.

Our analysis of Brighsquid projects at the file level revealed a consistent negative correlation between the number of bugs and the ownership metric for both time periods. This finding is in line with state-of-the-art research in the field. Furthermore, when comparing the results before and after March 2020, we observed a strengthening of the negative correlation between the number of bugs and the ownership metric, indicating that the assignment of user stories to developers strengthen the negative correlation between the number of bugs and the ownership metric at Brighsquid.

Table 3.4: Spearman correlation coefficients between metrics and the number of bug fixes on directory level.

| | March 2018 to March 2020 | | | | | March 2020 to March 2022 | | | | |
|--------------------------|--------------------------|--------------|--------------|--------------|-------------|--------------------------|--------------|--------------|--------------|-------------|
| Ownership Metrics | Proj1 | Proj2 | Proj3 | Proj4 | Avg. | Proj1 | Proj2 | Proj3 | Proj4 | Avg. |
| Avgownership | 0.219 | -0.121 | 0.026 | 0.136 | 0.065 | -0.292 | -0.046 | 0.101 | 0.108 | -0.032 |
| Ownershipdir | 0.054 | -0.124 | 0.126 | 0.136 | 0.048 | -0.041 | -0.032 | 0.155 | 0.113 | 0.049 |
| Minownerdir | 0.054 | -0.121 | 0.162 | 0.140 | 0.058 | -0.041 | -0.038 | 0.103 | 0.115 | 0.034 |
| Pcminors | 0.055 | -0.109 | 0.202 | 0.139 | 0.071 | -0.041 | -0.031 | 0.105 | 0.169 | 0.05 |
| Pcminimals | 0.044 | -0.156 | 0.130 | 0.125 | 0.035 | -0.037 | -0.035 | 0.102 | 0.122 | 0.038 |
| Pcmajors | 0.057 | -0.136 | -0.037 | 0.130 | 0.003 | -0.043 | -0.035 | 0.064 | 0.167 | 0.038 |
| Avgminors | 0.212 | -0.120 | 0.027 | 0.135 | 0.063 | -0.299 | -0.046 | 0.101 | 0.101 | -0.035 |
| Minownedfile | 0.141 | -0.052 | 0.177 | 0.171 | 0.109 | -0.170 | -0.047 | 0.144 | 0.429 | 0.089 |
| Avgminimals | 0.210 | -0.121 | 0.026 | 0.134 | 0.062 | -0.289 | -0.055 | 0.101 | 0.154 | -0.022 |
| Weakowneds | 0.355 | 0.110 | -0.060 | -0.148 | 0.064 | 0.363 | 0.046 | -0.179 | -0.181 | 0.012 |
| Aavgcontri butors | 0.361 | 0.251 | 0.441 | 0.27 | 0.33 | 0.175 | 0.244 | 0.516 | 0.37 | 0.32 |

Furthermore, we utilized a multiple linear regression model to investigate **RQ1** and analyze the influence of each ownership metric at the file level on the number of bugs. Our objective was to assess the relationship between ownership measures while accounting for source code characteristics. Additionally, we aimed to determine if these effects remained consistent or varied across the two time periods. We only did this evaluation at the file level, as the correlations for the directory level were nearly zero. To achieve this, we constructed five statistical models. We began with the “Base” model, which incorporated code metrics churn, size, and complexity. We then iteratively added ownership metrics one by one. We presented a comparison of the variance in bugs (\mathbb{R}^2) for the five statistical models during both time periods in Table 3.5. For both time periods, we observed that the inclusion of the Minor metric in our regression model resulted in the most significant improvement in explaining the variance in the number of bugs (18.8% before March 2020 and 4.5% after March 2020). On the other hand, the addition of the Major and Ownership metrics only led

Table 3.5: Variance in bugs for the Base model (Code metrics) and models with Minor, Major and Ownership added. An asterisk* denotes that a model showed models statistically significant improvement when the additional variable was added.

| | March 2018 to March 2020 | | | | |
|----------------------------------|--------------------------|------------------|----------------|---------------|-------------------|
| Model | Proj1 | Proj2 | Proj3 | Proj4 | Avg. |
| Base (Code metrics) | 41% | 98.1% | 31.1% | 85.5% | 63.9% |
| Base + Total | 82%*(+41.0%) | 99.7%*(+1.6%) | 70.0%*(+29.0%) | 88.9%*(+3.3%) | 85.1% (+18.7%) |
| Base + Minor | 83%*(+42.0%) | 99.7%*(+1.6%) | 70.8%*(+29.7%) | 87.6%*(+2.1%) | 85.2% (+18.8%) |
| Base + Minor + Major | 83% (+0%) | 99.7%(+0%) | 69.9%(-0.9%) | 88.6%*(+1.0%) | 85.3(+0%) |
| Base + Minor + Major + Ownership | 83% (+0%) | 99.7% (+0%) | 77.7%*(+7.8%) | 96.7%*(+8.1%) | 89.2% (+3.9%) |
| | March 2020 to March 2022 | | | | |
| Base (Code metrics) | 79% | 90.8% | 77.6% | 97.7% | 86.2% |
| Base + Total | 80%*(+1%) | 91.0%+(+0.2%) | 93.0*(+15.4%) | 98.9%*(+1.2%) | 90.7% (+4.4%) |
| Base + Minor | 98.9%*(+1.2%) | 91.0%(+0.2%) | 93.0%*(+15.4%) | 98.9%*(+1.2%) | 95.4% (+4.5%) |
| Base + Minor + Major | 98.9% (+0%) | 98.3%*(+7.3%) | 93.0%(+0%) | 98.9% (+0%) | 97.2% (+1.8%) |
| Base + Minor + Major + Ownership | 99.9%*(+1%) | 98.7% (+0.4%) | 93.0% (+0%) | 98.9% (+0%) | 97.6% (+0.3%) |

to marginal improvements in the variance and was statistically insignificant across all projects (3.9% before March 2020 and 0.3% after March 2020).

In Brightsquad, consistent with the findings in the literature, we observed a positive correlation between the number of minor contributors and the number of bugs. This correlation resulted in increased variance in the number of bugs when controlling for code metrics, both before and after the March 2020 period.

When comparing the Base models before and after March 2020, we observe a significant improvement in the explanatory power of code metrics for the variance of bugs. This improvement is consistent across all projects and, on average, the code metrics can now account for 86.2% of the variance in bugs. This represents a

notable enhancement of 22.3% compared to the pre-March 2020 period. Interestingly, we observed a considerable improvement in the resilience of projects after March 2020 toward ownership changes when controlling for code metrics. By changing the assignment process in March 2020, we observe that the differences between the Base model and the other four models are significantly lower (see Figure 3.5). When comparing the Base model with Base + Minor before March 2020, adding the Minor metric to the model increased the proportion of variance by 18.8% however, the addition of Minors accounts for only 4.5% increase post-March 2020.

The change in the assignment model in March 2020 effectively mitigated the impact of Minor contributors on the number of bugs when controlling for code metrics. This resulted in a reduction of 14.3% in the contribution of Minor contributors to the variance.

3.5.2 Bug predictability performance (RQ2)

Table 3.6: Details on Precision, Recall, and F-measure for predicting defective files.

| File level | | | | | | |
|-----------------|--------------------------|---------------|------------------|--------------------------|---------------|------------------|
| | March 2018 to March 2020 | | | March 2020 to March 2022 | | |
| ID | Precision | Recall | F-measure | Precision | Recall | F-measure |
| Proj1 | 0.55 | 0.40 | 0.46 | 0.67 | 0.65 | 0.66 |
| Proj2 | 0.53 | 0.38 | 0.44 | 0.45 | 0.40 | 0.43 |
| Proj3 | 0.44 | 0.65 | 0.52 | 0.69 | 0.65 | 0.67 |
| Proj4 | 0.70 | 0.63 | 0.66 | 0.77 | 0.73 | 0.75 |
| Avg. | 0.55 | 0.51 | 0.52 | 0.64 | 0.61 | 0.63 |
| Directory level | | | | | | |
| Proj1 | 0.70 | 0.55 | 0.61 | 0.66 | 0.65 | 0.65 |
| Proj2 | 0.67 | 0.61 | 0.64 | 0.68 | 0.68 | 0.68 |
| Proj3 | 0.71 | 0.65 | 0.68 | 0.79 | 0.75 | 0.77 |
| Proj4 | 0.69 | 0.61 | 0.65 | 0.71 | 0.69 | 0.70 |
| Avg. | 0.69 | 0.60 | 0.64 | 0.71 | 0.69 | 0.70 |

In Table 3.6 we provided a summary and comparison of the performance metrics (precision, recall, and F-measure) of the Random Forest classifier for classifying

defective sources at both the file and directory levels for Brightsquid projects.

At the file level, considering all four projects from March 2018 to March 2020 and March 2020 to March 2022, we achieved an average F-measure of 0.52 and 0.63, respectively. Moving on to the directory level, we attained an average F-measure of 0.64 and 0.70, respectively, for the same time periods. Analyzing the results presented in Table 3.6, it is evident that the Random Forest classifier performs better for identifying defective files and directories in the period from March 2020 to March 2022, compared to the period from March 2018 to March 2020, across all projects, with the exception of Proj2 at the file level.

The Random Forest model demonstrates improved precision and recall in predicting buggy files and directories following the change in the assignment model in 2020 at Brightsquid.

In line with the approach taken by Greiler et al. [68], we conducted a metric importance analysis to assess the predictive power of individual performance metrics at both the file and directory levels. Table 3.7 summarize our file-level observation and Table 3.8 summarizes the metric importance at the directory level. However, we only focused on the files and directories that have been changed during each time period. At the file level, the average importance scores for all ownership metrics increased for the four projects during the period from March 2020 to March 2022. Notably, Minors exhibited the highest predictive power for defective files, accounting

Table 3.7: Metric importance in accordance to ownership metrics across the changed files

| Project | March 2018 to March 2020 | | | | | March 2020 to March 2022 | | | | |
|---------|--------------------------|-------|--------|----------|-------|--------------------------|-------|--------|----------|-------|
| | Owner ship | Major | Minors | Minimals | Total | Owner ship | Major | Minors | Minimals | Total |
| Proj1 | 0.09 | 0.13 | 0.19 | 0.21 | 0.25 | 0.17 | 0.16 | 0.30 | 0.10 | 0.25 |
| Proj2 | 0.13 | 0.12 | 0.23 | 0.15 | 0.11 | 0.20 | 0.17 | 0.23 | 0.13 | 0.28 |
| Proj3 | 0.22 | 0.13 | 0.18 | 0.12 | 0.15 | 0.24 | 0.21 | 0.27 | 0.27 | 0.17 |
| Proj4 | 0.11 | 0.15 | 0.17 | 0.14 | 0.15 | 0.23 | 0.15 | 0.32 | 0.20 | 0.40 |
| Avg. | 0.14 | 0.13 | 0.17 | 0.14 | 0.15 | 0.21 | 0.17 | 0.28 | 0.17 | 0.27 |

Table 3.8: Metric importance in accordance to ownership metrics across the changed directories

| | March 2018 to March 2020 | | | | | March 2020 to March 2022 | | | | |
|-----------------|--------------------------|--------------|--------------|--------------|-------------|--------------------------|--------------|--------------|--------------|-------------|
| Metric | Proj1 | Proj2 | Proj3 | Proj4 | Avg. | Proj1 | Proj2 | Proj3 | Proj4 | Avg. |
| Avgownership | 0.10 | 0.17 | 0.22 | 0.27 | 0.19 | 0.17 | 0.18 | 0.34 | 0.12 | 0.20 |
| Ownershipdir | 0.24 | 0.10 | 0.33 | 0.10 | 0.19 | 0.21 | 0.13 | 0.27 | 0.08 | 0.17 |
| Minownerdir | 0.13 | 0.21 | 0.17 | 0.16 | 0.1 | 0.22 | 0.20 | 0.15 | 0.13 | 0.17 |
| Pcminors | 0.19 | 0.08 | 0.18 | 0.28 | 0.19 | 0.21 | 0.14 | 0.19 | 0.29 | 0.21 |
| Pcminimals | 0.05 | 0.09 | 0.05 | 0.12 | 0.17 | 0.09 | 0.14 | 0.08 | 0.15 | 0.11 |
| Pcmajors | 0.10 | 0.12 | 0.10 | 0.17 | 0.12 | 0.11 | 0.08 | 0.10 | 0.15 | 0.11 |
| Avgminors | 0.08 | 0.13 | 0.09 | 0.09 | 0.10 | 0.10 | 0.14 | 0.10 | 0.13 | 0.12 |
| Minownedfile | 0.14 | 0.13 | 0.18 | 0.19 | 0.16 | 0.15 | 0.13 | 0.17 | 0.22 | 0.17 |
| Avgminimals | 0.10 | 0.09 | 0.15 | 0.09 | 0.11 | 0.09 | 0.15 | 0.10 | 0.12 | 0.11 |
| Weakowneds | 0.12 | 0.12 | 0.10 | 0.17 | 0.12 | 0.10 | 0.13 | 0.09 | 0.09 | 0.10 |
| Avgcontributors | 0.13 | 0.21 | 0.08 | 0.09 | 0.13 | 0.09 | 0.19 | 0.08 | 0.07 | 0.11 |

for 17.5% before March 2020 and 27.8% after March 2020.

After the change in the assignment model, all file ownership metrics showed an increase in predictive power in and across all the projects. The number of Minor contributors is the metric with the highest importance score for both time periods.

Moving to the directory level, as shown in Table 3.8, our metric importance analysis indicated that the average percentage of commits by the highest contributor in a directory (Ownershipdir) possessed the highest average importance score (19.4%) for predicting defective source directories across all four projects from March 2018 to March 2020. During this period, the second most important metric on average was the average ownership value of a directory. However, after the process change in March 2020, the percentage of contributors with less than 50% commits (Modests) demonstrated the highest predictive power (20.9%) at the directory level. Analyzing the average importance scores of these directory-level metrics, we found that the scores of Avgownership, Pcminors, Pcmajors, Avgminors, Minownedfile, and Avgminimals increased after March 2020, while the importance scores of other metrics were higher prior to March 2020. Hence the results are inconclusive.

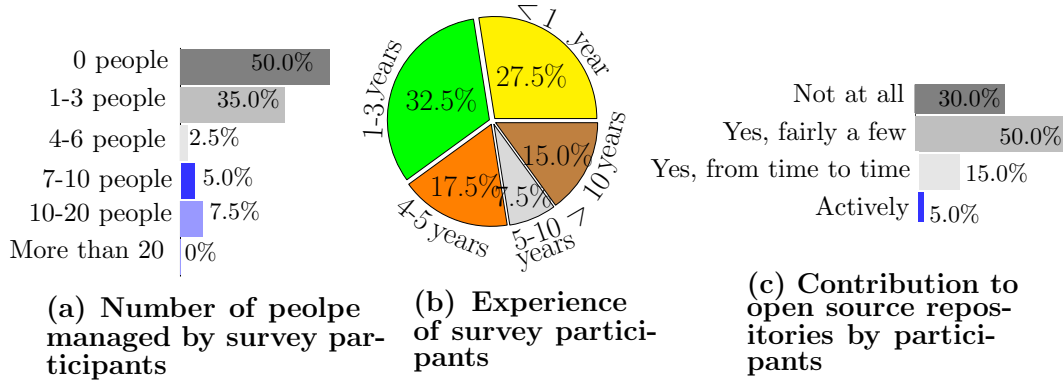


Figure 3.3: Demographics of survey participants

3.5.3 Survey with developers (RQ3)

A total of 67 developers participated in the survey, out of which 40 developers completed the survey. We discard the incomplete responses and present the results of the complete survey. 38 of the participants had the title of software developer in their team. The survey consisted of three demographic questions, the results of which are presented in Figure 3.3. 32.5% of the participants had one to three years of experience, 27.5% had less than one year of experience, and 15.0% had more than 10 years of experience. Regarding team management, 50.0% of the participants did not manage any developers in their team while 7.5% managed 10-20 developers. 70% of our participants were contributing to open-source projects.

Table 3.9 summarizes the questions and results regarding the source and degree of developers' sense of ownership. 85.0% of our participants expressed that they feel ownership toward the software artifacts. Among them, 20.95% felt most ownership toward code, 14.86% toward Tasks, and 13.51% toward bugs or issues.

When asked about the circumstances in which they experience the strongest sense of ownership for an artifact, 35% of our participants stated that it occurs when they are the sole authors of the artifact, while 22.5% mentioned feeling a heightened sense of ownership when they contribute more than others. Additionally, 17.5% of the developers reported feeling ownership based on their higher knowledge and expertise,

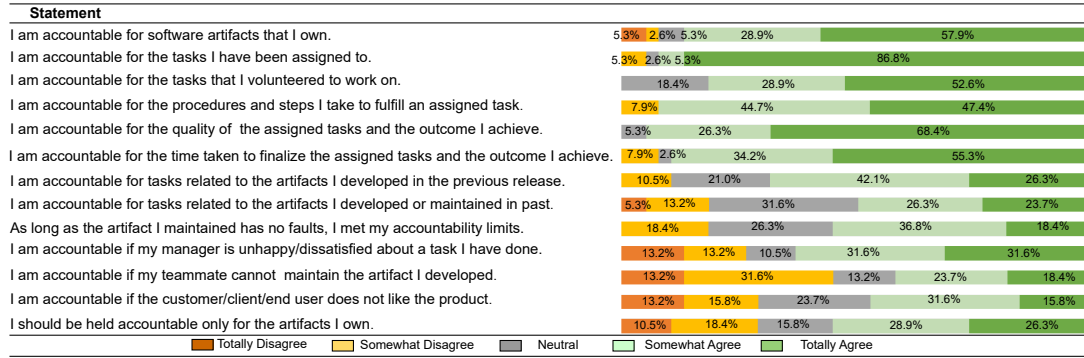


Figure 3.4: Accountability Levels of Developers in a Team. Participants' accountability were measured using a Likert scale.

while 12.5% attributed their sense of ownership to their historical contributions in maintaining the artifact. However, 60.0% of them indicated that they frequently or always perceive the ownership of artifacts as shared between themselves and others.















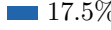
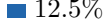
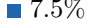
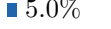
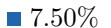
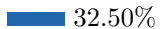

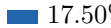
Developers mainly perceive ownership towards code and tasks, and they primarily determine their ownership based on their level of contribution, which accounts for the majority of cases (57.5%).

We provided the developers with a dictionary definition of accountability, which states that "Accountability refers to the real or perceived likelihood that the actions, decisions, or behaviors of an individual, group, or organization will be evaluated. It also entails the potential for the individual, group, or organization to receive rewards or sanctions based on this expected evaluation." After presenting this definition, we proceeded to ask the participants to identify the person towards whom they feel the highest level of accountability. Among developers, team leads and managers hold the highest level of accountability (84.2% of participants). Following that, developers feel accountable towards themselves (79%) and the developers express accountability towards their teammates and co-workers (73%) in the third place.

We conducted a survey in which we presented a series of 13 statements regarding accountability within their current software team. The developers were asked to indicate their level of agreement with each statement using a five-point Likert scale. The results, shown in Figure 3.4, revealed interesting insights.

A significant majority of participants, 94.7%, expressed agreement in feeling ac-

Table 3.9: Source and degree of developer's sense of ownership

| Do you feel ownership towards any software artifacts? | % of Participants |
|--|--|
| Yes |  85.0% |
| No |  15.0% |
| Which artifact do you feel most ownership towards? | % of Participants |
| Code |  20.95% |
| Task |  14.86% |
| Bug/Issue |  13.51% |
| Product |  9.46% |
| Test |  9.46% |
| User Story |  9.46% |
| Project |  8.78% |
| Requirement |  6.76% |
| Build |  5.41% |
| Others |  1.35% |
| When do you consider yourself an artifact owner? | % of Participants |
| All the artifact is written by me |  35.0% |
| I contributed more than others |  22.5% |
| I have the most knowledge and expertise on artifact |  17.5% |
| I was assigned to maintain it |  12.5% |
| Proposed the idea behind artifact (IP) |  7.5% |
| Initiated implementation |  5.0% |
| How often do you feel ownership of an artifact is shared between you and others? | % of Participants |
| Rarely |  7.50% |
| Sometimes |  32.50% |
| Frequently |  42.50% |
| Always |  17.50% |

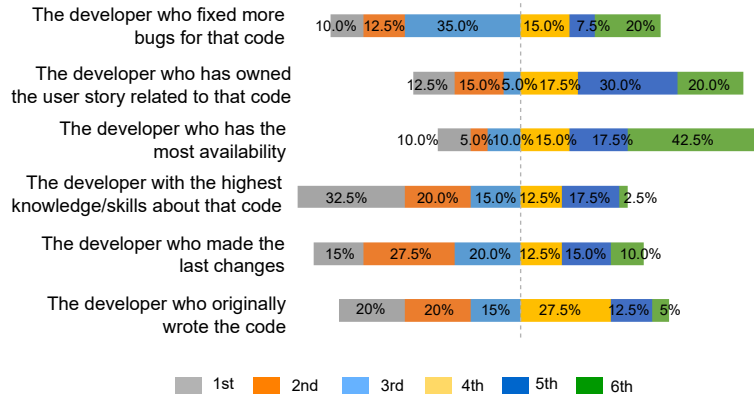


Figure 3.5: Who should be held accountable for maintaining an artifact with shared ownership?

countable for the quality of the outcome of the tasks assigned to them. Similarly, 92.1% of developers felt accountable for the procedure and steps taken to fulfill the assigned tasks. When it came to the tasks specifically assigned to them, 92.1% felt accountable without any disagreement. However, for tasks they volunteered to take, slightly fewer developers, 81.5%, expressed accountability with no disagreement. Moreover, 89.5% of developers showed a sense of accountability for timely task delivery. Interestingly, in the scenario where a teammate is unable to maintain an artifact developed by the participating developer, 44.8% of participants disagreed with feeling accountable for it, indicating a lower level of accountability in this particular situation.

Developers generally agreed on being accountable for the tasks assigned to them, as well as the time and steps taken to deliver them, and the quality of the outcomes.

Only 86.8% of developers feel accountable towards the artifacts they own. We then asked the developers to rank the individuals who should be held accountable for maintaining an artifact with shared ownership. The results of their ranking are presented in Figure 3.5. The majority, 52.5%, believe that developers with the highest knowledge and skills should be held accountable for maintaining a shared artifact. Additionally, 62.5% of the participants believe that the person who made the last changes to the file should be accountable for its maintenance, followed by 57.5% who

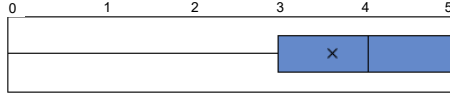


Figure 3.6: Developers’ perceived degree of relationship between accountability and ownership (5 = strongest degree).

believe that the developer with the highest number of bug fixes in a file should be held accountable for maintaining it.

When directly asked about the relationship between accountability and ownership (Figure 3.6), 5% of participants considered it a weak relationship (score of 2), while 67.5% considered it a strong or very strong relationship.

The majority of developers feel accountable towards their owned artifacts and perceive a significant relationship between accountability and ownership of these artifacts.

3.6 Discussion

Two years after transitioning from task assignments to assigning user stories to developers, the leadership team at Brightsquid has observed a positive impact on developers’ accountability, transforming the team culture from being solely focused on order-taking to fostering a culture of design and innovation:

“We have to get the team more involved in the kind of design, conversation leading to the solution that we want. In the current digital health ecosystem, we do not always have a defined product for a specific market. Therefore, our continuous effort is to encourage the market to adopt new solutions. This change our role from development management to vision management.”

While this change was inevitably driven by the increasing demand for digital health during the COVID-19 pandemic, the company has continued this practice due to the perceived higher level of accountability. Furthermore, our measurements have indicated improved code ownership across projects.

3.6.1 Assignment process and relation to accountability, ownership, and code quality

The sociotechnical aspects of software systems are inherently complex, and they contain intricate details that may have been overlooked in empirical investigations, making it challenging to establish causality. However, for this analysis, we have carefully selected four Brightsquad projects. The key distinguishing factor between the periods of 2018-2020 and 2020-2022 in these projects is the assignment model. Our selection is based on observations and has been agreed upon by the production team. Our investigation in **RQ1** and **RQ2** revealed that the change in the assignment process strengthened the correlation between ownership metrics and code quality factors. It also made code defects more predictable by using ownership metrics. These findings align with previous literature on the subject.

Our analysis at the directory level did not provide much insight into the changes when compared between the two time periods. However, through our analysis of Brightsquad projects at the file level, we discovered a consistent negative correlation between the number of bugs and the ownership metric for both time periods. This finding aligns with current research in the field. Additionally, when comparing the results before and after March 2020, we observed a strengthening of the negative correlation between the number of bugs and the ownership metric. When discussing this matter with the production team, it was perceived that the assignment of user stories to developers enhances their sense of accountability. This, in turn, ensures that the quality of the designed solution meets expectations, thereby reinforcing the negative correlation between bugs and the ownership metric at Brightsquad. In Brightsquad, the product manager does not specifically monitor ownership metrics, which include historical data on how developers have interacted with and modified a file. However, they interpret these findings as evidence that improved code quality and enhanced ownership are direct outcomes of the change in the assignment process. This change involves assigning user stories with greater delegation to the developers, making the

developer responsible for the overall outcome of the story rather than just a piece of code:

“Code quality, and I am not just talking about testing, but also considering acceptance criteria and peer reviews, depends upon the diligence of the process. The combination of people and process is what gives you the greatest likelihood of quality. Code ownership can vary on a sliding scale, and in my experience, the relevance of how many people touch the code becomes less significant when the process is well-defined and delegated appropriately.”

We observed a stronger correlation and predictive power between ownership metrics and the number of bugs at the file level (Table 3.3 and Table 3.5). Interestingly, when comparing the variance in the number of bugs controlled by code metrics (Table 3.5), we found:

First, the base model, which measures variance in the number of bugs based on code metrics, experienced a significant increase of 38% in enhanced variance. This indicates that code attributes alone can explain a considerable degree of variance in the number of bugs after the change in March 2020.

Second, the quality of all projects demonstrates greater resilience to change and ownership metrics. Previously, the addition of minor contributors could improve the variance in bugs by 18.8% compared to the base model. However, following the process change in 2020, this improvement was reduced to 4.5%.

We interpret this as the shift in the assignment model distributing ownership among all team members and shifting the mindset from code perfection to implementing feature sets and releasing the product. Consequently, the significance of who owns the code and their historical contributions in a file becomes less relevant in defining its quality.

“... along with engaging developers more in design they became more diligent about releasing the product and less about technical quality (to balance their efforts). This helped in reducing risk and adopting to the change during pandemic because we could see if the market adapts to what we have built.”

With the change in the assignment model, the team spends less effort on pre-defining the requirements which leads into identifying more requirement-related bugs after development;

“We do less diligence testing before a release hence the bugs can go up. We spent way less time on requirements hence there were more issues coming up because the features are less specified in advance. We became more reactive toward bugs and improvement requests.”

3.6.2 Survey Implications (RQ3)

When surveying the developers, it was found that 20.95% felt ownership towards code, while 14.86% felt ownership towards tasks. In comparison, only 9.46% perceived ownership towards user stories, and 6.76% towards requirements. Furthermore, when asked about accountability, the majority (52.7%) expressed controversial or neutral feelings towards being accountable for the satisfaction of customers or end users. This is whilst the survey showed a clear tendency of developers feeling accountable toward assigned tasks. When discussing these results with Brighsquid, the product manager stated:

“It all depends on the team culture. In Brightsquid, it is time to time from our team asking that what’s the task but usually we go around that by starting with a story, gathering the information like investigating the story present all the facts what is happening and then we talk with team on that story to decide the designer solution going forward.”

We observed a strong correlation of 0.82 between the participation of users in open-source development and their accountability toward end-user satisfaction. This suggests that the open-source mindset fosters a culture that encompasses a broader spectrum of accountability. In addition, we employed the Mann-Whitney test to assess potential differences in group means for the sense of accountability in a team (Figure 3.4) among open-source participants. Our analysis revealed that users with experience in open-source development feel significantly more accountable if their teammates cannot maintain the artifact they have developed ($p - value = 0.008$).

3.7 Threats to validity

This study should be interpreted within the limitations of being a case study for research questions **RQ1** and **RQ2** [45], [87]. Our focus was solely on the four most active projects within Brightsquad, which may restrict the generalizability of our findings to other organizations or open-source projects. However, it provides a fair evaluation of the situation within Brightsquad, as the majority of developers and resources are involved in these projects. We carefully observed all relevant factors before and after March 2020 to identify the roots of ownership changes, and no significant changes other than the assignment model were found. Nonetheless, as a sociotechnical system, the software team is complex, and there may be confounding factors that we did not identify. In our case study, we considered ownership metrics used in previous studies [6], [68], and it's worth noting that using different ownership metrics may yield different findings. We acknowledge a limitation of our study, as we focused solely on individual ownership levels and did not consider team ownership within Brightsquad. This limitation can impact the internal validity of our findings. To address this gap, future studies should include an examination of team ownership levels, as this may provide a more comprehensive understanding of how ownership impacts software development dynamics.

For the survey of **RQ3**, the conclusions drawn in our study are based on a survey,

which may introduce some inaccuracies and the number of participants is limited, the larger group of participants can make our conclusion more vigorous. Surveying software developers may not always capture a comprehensive perspective of real-world practices [87], [88]. However, surveys have been widely used as a research tool in software engineering. We view the survey as a complementary approach to other types of studies, such as mining, in order to gain a more holistic understanding of the research topic.

3.8 Related Work

While accountability has been discussed in various teams as a scientific discipline [89], [90], it has not been specifically studied within software teams. In this context, we provide a concise overview of related studies on accountability and ownership in software teams.

3.8.1 Individual and team accountability

Software development is a collaborative endeavor, and its social aspects have long been a topic of discussion. Many parallels can be drawn between the social behavior of human society and various aspects of software development. Research has explored the relationship between psychological ownership, self-identity, organizational accountability, a sense of belonging, and organizational citizenship [91]. Interestingly, psychological ownership has received limited attention in the field of software engineering [8], [92].

A sense of ownership inherently encompasses a sense of responsibility towards a target. This responsibility entails protecting and improving possessions and may involve controlling or limiting access by others [93]. On the other hand, accountability involves the expected rights and responsibilities of individuals, including the right to hold others accountable and the expectation of being held accountable oneself [94]. Feelings of psychological ownership are closely tied to attachment towards places,

objects, and people [93], [94]. The need for belongingness in the organization or workplace [95] is fulfilled when individuals feel like owners within the organization, satisfying their socio-emotional needs [96], [97].

3.8.2 Ownership in Software Teams

Ownership within software teams plays a pivotal role in fostering commitment, initiative, and delivering high-quality work [6], [7], [70], [98]. It encompasses two primary forms: Psychological Ownership, which relates to an individual's sense of possession [8], and Corporal Ownership, which pertains to the developmental history of software artifacts. Code ownership is widely recognized and facilitates accountability, task delegation, and identification of experts within a software product [53]. Research indicates that weak code ownership is associated with an increased likelihood of defects [6], [68], [75], [85]. In the agile context, task ownership is particularly important as it empowers team members to take responsibility for assigned tasks from task boards, thereby enhancing collaboration and project progress [19]. Bug ownership serves as a means to identify the responsible developers involved in bug-fixing efforts, whether it be the developer who introduced the code or another team member [65]. Additionally, test ownership designates individuals who contribute to specific tests within a test suite, streamlining test management and accountability [61]. Ownership and task assignment are closely interconnected [74], [75], [99]. Typically, the team lead assumes responsibility for task and bug assignments, investing significant time in determining the most suitable developer for each task [20]. Incorporating ownership considerations in task assignment enhances the efficiency and effectiveness of bug resolution within the team [24]–[26], [100]. Several systematic literature reviews have provided comprehensive insights into the relationship between code ownership, quality, and task assignments [22], [101], [102].

3.9 Conclusion

Brighsquid implemented a new task assignment process aimed at enhancing developer accountability toward customers. The findings from both code change analysis and a survey of 67 developers demonstrate the positive impact of increased accountability on code ownership. Notably, the revised assignment model not only heightened the perceived sense of accountability within the production team but also improved the resilience of the code against ownership changes. Moreover, when comparing the data before and after March 2020, there was a notable reinforcement of the negative correlation between the number of bugs and the ownership metric. This observation suggests that the assignment of user stories played a significant role in reducing the occurrence of bugs. The survey results revealed an intriguing statistic, with 67.5% of developers associating perceived accountability with artifact ownership. This empirical examination sheds light on accountability within software teams, opening up new avenues for further evaluation. By understanding the impact of accountability on ownership, future research can focus on developing effective strategies to foster motivation and a sense of belonging within teams, ultimately leading to increased productivity.

Chapter 4

Conclusion and Future Work

This thesis focuses on the various aspects of ownership and accountability within software teams and their relationships. Motivated by the problem in our partner company, we primarily focused on investigating the influence of ownership and accountability on code quality in two separate case studies (2 and 3) and across overall nine projects of Brightsquad. We further complemented our study with a survey to gather insights and perspectives from software developers. The scope of our study was constrained to active projects within our partner company. Consequently, the findings derived from our research may not be generalizable to projects of other organizations, particularly those in the open-source domain. It is also essential to acknowledge that the development processes can vary across various projects in different companies. As a result, the impact of ownership and accountability on code quality may differ significantly from what we observed in our case study.

4.1 Conclusion

In Chapter 2, we conducted a systematic literature review to identify existing ownership models. As a result of the systematic gathering of the studies, we analyzed 79 papers published between 2005 and 2022. Our study established a taxonomy of ownership artifacts, examined modeling variables, and assessed the replication status of each study. We found that code ownership emerged as the most widely discussed

artifact in the software industry. Furthermore, in collaboration with our partner company Brightsquad, we replicated three different studies from Bird et al. [6], Greiler et al.[68], and Foucault et al. [75]. Using Brightsquad data, we compared their code ownership status with the reported state-of-the-art and practice. This analysis contributes to a better understanding of code ownership and its implications for software development.

Further, we deepened this study by analyzing the impact of task assignment on the code ownership. In Chapter 3, we investigated the impact of a new task assignment process implemented at Brightsquad, focusing on enhancing developer accountability towards customers. In March 2020 and with the emergence of COVID-19 pandemic, Brightsquad changed their task assignment process for enhanced accountability. To understand the impact of task assignment on developers' accountability, we performed code change analysis using Brightsquad data before and after March 2020. The results indicate that the revised model enhanced both the perceived sense of accountability within the production team and the code's ability to handle ownership changes effectively. We also did a survey with software developers to understand the relationship between ownership and accountability in other software teams. The survey findings revealed that a significant majority of developers (67.5%) associated perceived accountability with artifact ownership. These findings highlight the positive relationship between accountability and ownership in software development and suggest the importance of fostering a culture of accountability for improved code quality and team productivity.

4.2 Future Work

In this thesis, we focused on only code ownership metrics, specifically individual code ownership. Software engineering research includes a wide spectrum of ownership definitions and metrics (see Chapter 2). In the future, it would be beneficial to expand the analysis beyond individual code ownership and explore the impact of

organizational code ownership on code quality. Investigating other ownership artifacts such as tasks, tests, bugs, and requirements could provide further insights into their influence on software companies.

In this study, we focused on bug prediction using the Random Forest model, as it was consistent with previous studies (Greiler et al., 2015). However, future research can explore the performance of other classification models to compare their effectiveness in bug prediction. By evaluating and comparing multiple models, we can gain insights into which approach yields better results. Furthermore, our analysis involved running the Random Forest model separately for different projects within Brightsquad. In the future, it would be beneficial to combine the ownership metrics data from all considered projects and apply the classification model to assess if the overall performance improves. This approach would provide a more comprehensive understanding of the relationship between ownership and bug prediction across the entire organization. Additionally, to further enrich our analysis, it would be valuable to consider additional metrics such as developer experience (Rahman et al., 2011), bug-fixing time (Zhu et al., 2021), and the number of lines of code required to fix bugs (Rahman et al., 2011). Including these metrics can provide a more comprehensive view of the factors influencing bug prediction and ownership dynamics within the software development process.

In our exploration of the relationship between ownership and accountability, we focused primarily on Brightsquad through the case study and surveyed other software developers. However, it is important to acknowledge that there are other measures that can impact the relationship between ownership and accountability, such as autonomy, responsibility, organizational size and culture, developer incentives, and performance rewards. By considering these additional measures, we can gain a more comprehensive understanding of how ownership and accountability interact and influence each other within the software development context. These factors may play a significant role in shaping individuals' sense of ownership, their willingness to take

responsibility for their work, and their overall accountability within the organization. To gain a more comprehensive understanding of the relationship between ownership and accountability, it is possible to conduct separate surveys on each concept and then analyze the correlation between them. This approach allows for focused exploration of the dimensions and factors related to ownership and accountability individually.

Furthermore, further research can involve analyzing different teams and companies, including open-source projects, to determine if the impact of ownership is consistent or varies. The size, the culture, the geographical distribution, the life cycle management method can be controlled in these further experimentation to evaluate if and to what extent accountability and ownership are interconnected in different circumstances. It would be valuable to further explore how different teams increase their developer's productivity and improve team culture within their organization and how it affects the ownership of different artifacts. Conducting surveys with a larger number of developers would provide diverse perspectives and valuable advice.

Overall, the insights derived from this study can provide valuable guidance for establishing best practices in software development teams and assist in making the right decisions regarding code ownership and accountability. By understanding the impact of accountability on ownership, organizations will gain clarity on the metrics they should prioritize to enhance developers' sense of ownership and accountability. Additionally, this study will help future researchers to understand how accountability influences ownership in improving collaboration and productivity within teams in software development.

Bibliography

- [1] S. J. Kabeer, M. Nayebi, G. Ruhe, C. Carlson, and F. Chew, “Predicting the vector impact of change-an industrial case study at brightsquid,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, 2017, pp. 131–140.
- [2] M. Nayebi, S. J. Kabeer, G. Ruhe, C. Carlson, and F. Chew, “Hybrid labels are the new measure!” *IEEE Software*, vol. 35, no. 1, pp. 54–57, 2017.
- [3] M. Nayebi, L. Dicke, R. Ittyipe, C. Carlson, and G. Ruhe, “Essmart way to manage customer requests,” *Empirical Software Engineering*, vol. 24, pp. 3755–3789, 2019.
- [4] M. Nayebi, Y. Cai, R. Kazman, *et al.*, “A longitudinal study of identifying and paying down architecture debt,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2019, pp. 171–180.
- [5] A. Hemmati, C. Carlson, M. Nayebi, G. Ruhe, and C. Saunders, “Analysis of software service usage in healthcare communication services,” in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, 2017, pp. 565–566.
- [6] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, “Don’t touch my code! examining the effects of ownership on software quality,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.
- [7] L. Hattori and M. Lanza, “Mining the history of synchronous changes to refine code ownership,” in *2009 6th iee international working conference on mining software repositories*, IEEE, 2009, pp. 141–150.
- [8] T. (Chung, P. N. Sharma, and S. L. Daniel, “The impact of person-organization fit and psychological ownership on turnover in open source software projects.,” in *AMCIS*, 2015.
- [9] Y. Dubinsky, A. Yaeli, and A. Kofman, “Effective management of roles and responsibilities: Driving accountability in software development teams,” *IBM Journal of Research and Development*, vol. 54, no. 2, pp. 4–1, 2010.

- [10] R. Burga, C. Spraakman, C. Balestreri, and D. Rezaia, “Examining the transition to agile practices with information technology projects: Agile teams and their experience of accountability,” *International Journal of Project Management*, vol. 40, no. 1, pp. 76–87, 2022.
- [11] O. McHugh, K. Conboy, and M. Lang, “Agile practices: The impact on trust in software project teams,” *IEEE software*, vol. 29, no. 3, pp. 71–76, 2011.
- [12] A. Trendowicz and J. Münch, “Factors influencing software development productivity—state-of-the-art and industrial experiences,” *Advances in computers*, vol. 77, pp. 185–241, 2009.
- [13] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, “The space of developer productivity: There’s more to it than you think,” *Queue*, vol. 19, no. 1, pp. 20–48, 2021.
- [14] G. Sudhakar, A. Farooq, and S. Patnaik, “Measuring productivity of software development teams,” *Serbian Journal of Management*, vol. 7, no. 1, pp. 65–75, 2012.
- [15] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, “Software developers’ perceptions of productivity,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 19–29.
- [16] L. Bao, T. Li, X. Xia, K. Zhu, H. Li, and X. Yang, “How does working from home affect developer productivity?—a case study of baidu during the covid-19 pandemic,” *Science China Information Sciences*, vol. 65, no. 4, p. 142 102, 2022.
- [17] J. Warsta and P. Abrahamsson, “Is open source software development essentially an agile method,” in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Citeseer, 2003, pp. 143–147.
- [18] K. H. Judy and I. Krumins-Beens, “Great scrums need great product owners: Unbounded collaboration and collective product ownership,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, IEEE, 2008, pp. 462–462.
- [19] Y. Shastri, R. Hoda, and R. Amor, “Understanding the roles of the manager in agile project management,” in *Proceedings of the 10th Innovations in Software Engineering Conference*, 2017, pp. 45–55.
- [20] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?” In *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.
- [21] Y. Zhen, A. Khan, S. Nazir, Z. Huiqi, A. Alharbi, and S. Khan, “Crowdsourcing usage, task assignment methods, and crowdsourcing platforms: A systematic literature review,” *Journal of Software: Evolution and Process*, vol. 33, no. 8, e2368, 2021.

- [22] S. Mahmood, S. Anwer, M. Niazi, M. Alshayeb, and I. Richardson, “Key factors that influence task allocation in global software development,” *Information and Software Technology*, vol. 91, pp. 102–122, 2017.
- [23] Y. Bugayenko, A. Bakare, A. Cheverda, *et al.*, “Prioritizing tasks in software development: A systematic literature review,” *Plos one*, vol. 18, no. 4, e0283838, 2023.
- [24] O. Baysal, M. W. Godfrey, and R. Cohen, “A bug you like: A framework for automated assignment of bugs,” in *2009 IEEE 17th International Conference on Program Comprehension*, IEEE, 2009, pp. 297–298.
- [25] J. Anvik, “Automating bug report assignment,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 937–940.
- [26] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, “How long will it take to fix this bug?” In *Fourth International Workshop on Mining Software Repositories (MSR’07: ICSE Workshops 2007)*, IEEE, 2007, pp. 1–1.
- [27] D. Budgen and P. Brereton, “Performing systematic literature reviews in software engineering,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 1051–1052.
- [28] H. Yang, “Total cost of ownership for application replatform by open-source sw,” *Procedia Computer Science*, vol. 91, pp. 677–682, 2016.
- [29] S. Jansen and W. Rijsemus, “Reducing customers’ total cost of ownership within a software supply network,” in *2006 22nd IEEE International Conference on Software Maintenance*, IEEE, 2006, pp. 269–271.
- [30] L. Fink, J. Shao, Y. Lichtenstein, and S. Haeffiger, “The ownership of digital infrastructure: Exploring the deployment of software libraries in a digital innovation cluster,” *Journal of Information Technology*, vol. 35, no. 3, pp. 251–269, 2020.
- [31] E. Kerfoot and S. McKeever, “Deadlock freedom through object ownership,” 2009.
- [32] S. Negara, R. K. Karmani, and G. Agha, “Inferring ownership transfer for efficient message passing,” *ACM SIGPLAN Notices*, vol. 46, no. 8, pp. 81–90, 2011.
- [33] N. Cameron and W. Dietl, *Comparing universes and existential ownership types*. Citeseer, 2009.
- [34] A. Taivalsaari, T. Mikkonen, and K. Systä, “Liquid software manifesto: The era of multiple device ownership and its implications for software architecture,” in *2014 IEEE 38th Annual Computer Software and Applications Conference*, IEEE, 2014, pp. 338–343.
- [35] J. Hur, D. Koo, Y. Shin, and K. Kang, “Secure data deduplication with dynamic ownership management in cloud storage,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 11, pp. 3113–3125, 2016.

- [36] R. Kaur, S. Singh, and H. Kumar, “An intrinsic authorship verification technique for compromised account detection in social networks,” *Soft Computing*, vol. 25, no. 6, pp. 4345–4366, 2021.
- [37] J. Huh, M. W. Newman, and M. S. Ackerman, “Supporting collaborative help for individualized use,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 3141–3150.
- [38] D. Rayside and L. Mendel, “Object ownership profiling: A technique for finding and fixing memory leaks,” in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 194–203.
- [39] T. Ravitch and B. Liblit, “Analyzing memory ownership patterns in c libraries,” in *Proceedings of the 2013 international symposium on memory management*, 2013, pp. 97–108.
- [40] C. Zhao, W. Song, X. Liu, L. Liu, and X. Zhao, “Research on authorship attribution of article fragments via rnns,” in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, IEEE, 2018, pp. 156–159.
- [41] M. Narayanan, J. Gaston, G. Dozier, *et al.*, “Adversarial authorship, sentiment analysis, and the authorweb zoo,” in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2018, pp. 928–932.
- [42] J. Ji, S. Park, G. Woo, and H. Cho, “Understanding the evolution process of program source for investigating software authorship and plagiarism,” in *2007 2nd International Conference on Digital Information Management*, IEEE, vol. 1, 2007, pp. 98–103.
- [43] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood, “Replication of experimental results in software engineering,” *International Software Engineering Research Network (ISERN) Technical Report ISERN-96-10*, University of Strathclyde, vol. 2, 1996.
- [44] O. S. Gómez, N. Juristo, and S. Vegas, “Understanding replication of experiments in software engineering: A classification,” *Information and Software Technology*, vol. 56, no. 8, pp. 1033–1048, 2014.
- [45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [46] N. Juristo and S. Vegas, “The role of non-exact replications in software engineering experiments,” *Empirical Software Engineering*, vol. 16, no. 3, pp. 295–324, 2011.
- [47] B. Tessem, “Individual empowerment of agile and non-agile software developers in small teams,” *Information and software technology*, vol. 56, no. 8, pp. 873–889, 2014.

- [48] M. Nayebe, G. Ruhe, and T. Zimmermann, “Mining treatment-outcome constructs from sequential software engineering data,” *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 393–411, 2019.
- [49] J. Businge, S. Kawuma, E. Bainomugisha, F. Khomh, and E. Nabaasa, “Code authorship and fault-proneness of open-source android applications: An empirical study,” in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 33–42.
- [50] C. Müller, G. Reina, and T. Ertl, “In-situ visualisation of fractional code ownership over time,” in *Proceedings of the 8th International Symposium on Visual Information Communication and Interaction*, 2015, pp. 13–20.
- [51] E. Bogomolov, V. Kovalenko, Y. Rebryk, A. Bacchelli, and T. Bryksin, “Authorship attribution of source code: A language-agnostic approach and applicability in software engineering,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 932–944.
- [52] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, “Revisiting code ownership and its relationship with software quality in the scope of modern code review,” in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1039–1050.
- [53] F. Rahman and P. Devanbu, “Ownership, experience and defects: A fine-grained study of authorship,” in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 491–500.
- [54] N. Nagappan, B. Murphy, and V. Basili, “The influence of organizational structure on software quality,” in *2008 ACM/IEEE 30th International Conference on Software Engineering*, IEEE, 2008, pp. 521–530.
- [55] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat, “Mining software repositories for accurate authorship,” in *2013 IEEE international conference on software maintenance*, IEEE, 2013, pp. 250–259.
- [56] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, “An analysis of the effect of code ownership on software quality across windows, eclipse, and firefox,” *Technical report, University of California*, 2010.
- [57] Y. Shastri, R. Hoda, and R. Amor, “The role of the project manager in agile software development projects,” *Journal of Systems and Software*, vol. 173, p. 110 871, 2021.
- [58] E. Dutra, B. Diirr, and G. Santos, “Human factors and their influence on software development teams-a tertiary study,” in *Brazilian Symposium on Software Engineering*, 2021, pp. 442–451.
- [59] S. Datta, “Perspectives on task ownership in mobile operating system development (invited talk),” in *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*, 2014, pp. 11–12.

- [60] G. D. S. Hadad, J. H. Doorn, and J. C. S. do Prado Leite, “Requirements authorship: A family process pattern,” in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, IEEE, 2017, pp. 248–251.
- [61] K. Herzig and N. Nagappan, “The impact of test ownership and team structure on the reliability and effectiveness of quality test runs,” in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.
- [62] B. Caglayan and A. Bener, “Issue ownership activity in two large software projects,” *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–7, 2012.
- [63] D. Bertram, A. Voida, S. Greenberg, and R. Walker, “Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams,” in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, 2010, pp. 291–300.
- [64] J. Anvik and G. C. Murphy, “Reducing the effort of bug report triage: Recommenders for development-oriented decisions,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, pp. 1–35, 2011.
- [65] W. Zhu and M. W. Godfrey, “Mea culpa: How developers fix their own simple bugs differently from other developers,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, IEEE, 2021, pp. 515–519.
- [66] S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan, “An empirical study of build maintenance effort,” in *2011 33rd International Conference on Software Engineering (ICSE)*, IEEE, 2011, pp. 141–150.
- [67] M. Shridhar, B. Adams, and F. Khomh, “A qualitative analysis of software build system changes and build ownership styles,” in *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*, 2014, pp. 1–10.
- [68] M. Greiler, K. Herzig, and J. Czerwonka, “Code ownership and software quality: A replication study,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, IEEE, 2015, pp. 2–12.
- [69] A. Meneely, P. Rotella, and L. Williams, “Does adding manpower also affect quality? an empirical, longitudinal analysis,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 81–90.
- [70] L. M. Maruping, X. Zhang, and V. Venkatesh, “Role of collective ownership and coding standards in coordinating expertise in software project teams,” *European Journal of Information Systems*, vol. 18, no. 4, pp. 355–371, 2009.
- [71] S. Kaisler, F. Armour, and J. A. Espinosa, “Introduction to big data: Challenges, opportunities, and realities minitrack,” in *2014 47th Hawaii International Conference on System Sciences*, IEEE, 2014, pp. 728–728.

- [72] R. Wendler, “The maturity of maturity model research: A systematic mapping study,” *Information and software technology*, vol. 54, no. 12, pp. 1317–1339, 2012.
- [73] M. L. Drury-Grogan, K. Conboy, and T. Acton, “Examining decision characteristics & challenges for agile software development,” *Journal of Systems and Software*, vol. 131, pp. 248–265, 2017.
- [74] M. Foucault, J.-R. Falleri, and X. Blanc, “Code ownership in open-source software,” in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–9.
- [75] M. Foucault, C. Teyton, D. Lo, X. Blanc, and J.-R. Falleri, “On the usefulness of ownership metrics in open-source software projects,” *Information and Software Technology*, vol. 64, pp. 102–112, 2015.
- [76] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, “The case for context-driven software engineering research: Generalizability is overrated,” *IEEE Software*, vol. 34, no. 5, pp. 72–75, 2017.
- [77] T. Menzies, A. Butcher, D. Cok, *et al.*, “Local versus global lessons for defect prediction and effort estimation,” *IEEE Transactions on software engineering*, vol. 39, no. 6, pp. 822–834, 2012.
- [78] C. I. Bezerra, J. C. de Souza Filho, E. F. Coutinho, *et al.*, “How human and organizational factors influence software teams productivity in covid-19 pandemic: A brazilian survey,” in *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, 2020, pp. 606–615.
- [79] C. Miller, P. Rodeghero, M.-A. Storey, D. Ford, and T. Zimmermann, ““ how was your weekend?” software development teams working from home during covid-19,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021, pp. 624–636.
- [80] D. Russo, P. H. Hanel, S. Altnickel, and N. Van Berkel, “The daily life of software engineers during the covid-19 pandemic,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2021, pp. 364–373.
- [81] D. Ford, M.-A. Storey, T. Zimmermann, *et al.*, “A tale of two cities: Software developers working from home during the covid-19 pandemic,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–37, 2021.
- [82] R. Hoda and L. K. Murugesan, “Multi-level agile project management challenges: A self-organizing team perspective,” *Journal of Systems and Software*, vol. 117, pp. 245–257, 2016.
- [83] R. Hoda and J. Noble, “Becoming agile: A grounded theory of agile transitions in practice,” in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, IEEE, 2017, pp. 141–151.

- [84] M. E. Nordberg, “Managing code ownership,” *IEEE software*, vol. 20, no. 2, pp. 26–33, 2003.
- [85] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, “Does distributed development affect software quality? an empirical case study of windows vista,” *Communications of the ACM*, vol. 52, no. 8, pp. 85–93, 2009.
- [86] S. L. Pfleeger and B. A. Kitchenham, “Principles of survey research: Part 1: Turning lemons into lemonade,” *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 6, pp. 16–18, 2001.
- [87] D. Lo, N. Nagappan, and T. Zimmermann, “How practitioners perceive the relevance of software engineering research,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 415–425.
- [88] T. Dyba, B. A. Kitchenham, and M. Jorgensen, “Evidence-based software engineering for practitioners,” *IEEE software*, vol. 22, no. 1, pp. 58–65, 2005.
- [89] J. R. Katzenbach and D. K. Smith, “The discipline of teams,” *Harvard business review*, vol. 83, no. 7, p. 162, 2005.
- [90] L. M. Marx and F. Squintani, “Individual accountability in teams,” *Journal of Economic Behavior & Organization*, vol. 72, no. 1, pp. 260–273, 2009.
- [91] I. Buchem, “Psychological ownership and personal learning environments: Do sense of ownership and control really matter,” in *PLE Conference Proceedings*, Citeseer, vol. 1, 2012.
- [92] T. Chung, P. Sharma, and D. Sherae, “The impact of person-organization fit and psychological ownership on turnover in open source software projects,” 2015.
- [93] J. B. Avey, B. J. Avolio, C. D. Crossley, and F. Luthans, “Psychological ownership: Theoretical extensions, measurement and relation to work outcomes,” *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior*, vol. 30, no. 2, pp. 173–191, 2009.
- [94] K. S. Corts, “Teams versus individual accountability: Solving multitask problems through job design,” *The RAND Journal of Economics*, vol. 38, no. 2, pp. 467–479, 2007.
- [95] J. Shakespeare-Finch and E. Daley, “Workplace belongingness, distress, and resilience in emergency service workers,” *Psychological Trauma: Theory, Research, Practice, and Policy*, vol. 9, no. 1, p. 32, 2017.
- [96] A. T. Hall, M. G. Bowen, G. R. Ferris, M. T. Royle, and D. E. Fitzgibbons, “The accountability lens: A new way to view management issues,” *Business Horizons*, vol. 50, no. 5, pp. 405–413, 2007.
- [97] H. Nissenbaum, “Computing and accountability,” *Communications of the ACM*, vol. 37, no. 1, pp. 72–81, 1994.

- [98] T. Sedano, P. Ralph, and C. Péraire, "Practice and perception of team code ownership," in *Proceedings of the 20th international conference on evaluation and assessment in software engineering*, 2016, pp. 1–6.
- [99] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "'not my bug!' and other reasons for software bug report reassignments," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, 2011, pp. 395–404.
- [100] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, IEEE, 2007, pp. 2–2.
- [101] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.
- [102] A. Niknafs, J. Denzinger, and G. Ruhe, "A systematic literature review of the personnel assignment problem," in *Proceedings of the International Multiconference of Engineers and Computer Scientists, Hong Kong*, 2013.

Appendix (I) Selected studies

- [S1] Maruping, Likoebe M., Xiaojun Zhang, and Viswanath Venkatesh. "Role of collective ownership and coding standards in coordinating expertise in software project teams." *European Journal of Information Systems* 18.4 (2009): 355-371.
- [S2] Judy, Ken H., and Ilio Krumins-Beens. "Great scrums need great product owners: Unbounded collaboration and collective product ownership." *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE, 2008.
- [S3] Nagappan, Nachiappan, Brendan Murphy, and Victor Basili. "The influence of organizational structure on software quality." *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008.
- [S4] Hattori, Lile, and Michele Lanza. "Mining the history of synchronous changes to refine code ownership." *2009 6th IEEE international working conference on mining software repositories*. IEEE, 2009.
- [S5] Mockus, Audris. "Succession: Measuring transfer of code and developer productivity." *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009.
- [S6] Bird, Christian, et al. "Does distributed development affect software quality? an empirical case study of windows vista." *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009.
- [S7] Bird, Christian, et al. "Don't touch my code! Examining the effects of ownership on software quality." *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011.
- [S8] Rahman, Foyzur, and Premkumar Devanbu. "Ownership, experience and defects: a fine-grained study of authorship." *Proceedings of the 33rd International Conference on Software Engineering*. 2011.

- [S9] Caglayan, Bora, and Ayse Bener. "Issue ownership activity in two large software projects." *ACM SIGSOFT Software Engineering Notes* 37.6 (2012): 1-7.
- [S10] Corley, Christopher S., Elizabeth A. Kammer, and Nicholas A. Kraft. "Modeling the ownership of source code topics." 2012 20th IEEE International Conference on Program Comprehension (ICPC). IEEE, 2012.
- [S11] Bird, Christian, and Nachiappan Nagappan. "Who? where? what? examining distributed development in two large open source projects." 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 2012.
- [S12] Tennyson, Matthew F. "A replicated comparative study of source code authorship attribution." 2013 3rd International Workshop on Replication in Empirical Software Engineering Research. IEEE, 2013.
- [S13] Meng, Xiaozhu, et al. "Mining software repositories for accurate authorship." 2013 IEEE international conference on software maintenance. IEEE, 2013.
- [S14] Datta, Subhajit. "Perspectives on task ownership in mobile operating system development (invited talk)." *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*. 2014.
- [S15] Foucault, Matthieu, Jean-Rémy Falleri, and Xavier Blanc. "Code ownership in open-source software." *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. 2014.
- [S16] Greiler, Michaela, Kim Herzig, and Jacek Czerwonka. "Code ownership and software quality: A replication study." 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, 2015.
- [S17] Müller, Christoph, Guido Reina, and Thomas Ertl. "In-situ visualisation of fractional code ownership over time." 8th International Symposium on Visual Information Communication and Interaction. 2015.
- [S18] Qiu, Yilin, et al. "An empirical study of developer quality." 2015 IEEE International Conference on Software Quality, Reliability and Security-Companion. IEEE, 2015.
- [S19] Foucault, Matthieu, et al. "On the usefulness of ownership metrics in open-source software projects." *Information and Software Technology* 64 (2015): 102-112.
- [S20] Faragó, Csaba, Péter Hegedűs, and Rudolf Ferenc. "Code ownership: Impact on maintainability." *International Conference on Computational Science and Its Applications*. Springer, Cham, 2015.
- [S21] Thongtanunam, Patanamon, et al. "Revisiting code ownership and its relationship with software quality in the scope of modern code review." *Proceedings of the 38th international conference on software engineering*. 2016.
- [S22] Lindsjörn, Yngve, et al. "Teamwork quality and project success in software development: A survey of agile development teams." *Journal of Systems and Software* 122 (2016): 274-286.
- [S23] Businge, John, et al. "Code authorship and fault-proneness of open-source android applications: An empirical study." *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. 2017.
- [S24] Zhu, Wenhan, and Michael W. Godfrey. "Mea culpa: How developers fix their own simple bugs differently from other developers." *arXiv preprint arXiv:2103.11894* (2021).
- [S25] Herzig, Kim, and Nachiappan Nagappan. "The impact of test ownership and team structure on the reliability and effectiveness of quality test runs." *Proceedings of the 8th*

ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2014.

[S26] Meneely, Andrew, Pete Rotella, and Laurie Williams. "Does adding manpower also affect quality? an empirical, longitudinal analysis." Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011.

[S27] Foucault, Matthieu, et al. "Impact of developer turnover on quality in open-source software." Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015.

[S28] Shridhar, Mini, Bram Adams, and Foutse Khomh. "A qualitative analysis of software build system changes and build ownership styles." Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement. 2014.

[S29] Shimagaki, Junji, et al. "A study of the quality-impacting practices of modern code review at sony mobile." Proceedings of the 38th International Conference on Software Engineering Companion. 2016.

[S30] Orru, Matteo, and Michele Marchesi. "A case study on the relationship between code ownership and refactoring activities in a Java software system." 2016 IEEE/ACM 7th International Workshop on Emerging Trends in Software Metrics (WETSoM). IEEE, 2016.

[S31] Vasilescu, Bogdan, et al. "Quality and productivity outcomes relating to continuous integration in GitHub." Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015.

[S32] Chung, Tingting, Pratyush Sharma, and Daniel Sherae. "The impact of person-organization fit and psychological ownership on turnover in open source software projects." (2015).

[S33] Çaglayan, Bora, and Ayşe Başar Bener. "Effect of developer collaboration activity on software quality in two large scale projects." Journal of Systems and Software 118 (2016): 288-296.

[S34] Jabangwe, Ronald, Darja Šmite, and Emil Hessbo. "Distributed software development in an offshore outsourcing project: A case study of source code evolution and quality." Information and Software Technology 72 (2016): 125-136.

[S35] Bogomolov, Egor, et al. "Authorship attribution of source code: A language-agnostic approach and applicability in software engineering." Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021.

[S36] Gull, Muqaddas, Tehseen Zia, and Muhammad Ilyas. "Source code author attribution using author's programming style and code smells." International Journal of Intelligent Systems and Applications 9.5 (2017): 27.

[S37] Di Penta, Massimiliano, and Daniel M. German. "Who are source code contributors and how do they change?." 2009 16th Working Conference on Reverse Engineering. IEEE, 2009.

[S38] Hadad, Graciela Dora Susana, Jorge Horacio Doorn, and Julio Cesar Sampaio do Prado Leite. "Requirements authorship: a family process pattern." 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). IEEE, 2017.

[S39] Dauber, Edwin, et al. "Git blame who? stylistic authorship attribution of small, incomplete source code fragments." Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings. 2018.

- [S40] Zeidman, Robert. "Software source code correlation." 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSTAR'06). IEEE, 2006.
- [S41] Ullah, Farhan, et al. "Source code authorship attribution using hybrid approach of program dependence graph and deep learning model." IEEE Access 7 (2019): 141987-141999.
- [S42] Zhao, Jie, Guohua Zhan, and Juan Feng. "Disputed authorship in C program code after detection of plagiarism." 2008 International Conference on Computer Science and Software Engineering. Vol. 1. IEEE, 2008.
- [S43] Gong, Siyi, and Hao Zhong. "Code authors hidden in file revision histories: An empirical study." 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC). IEEE, 2021.
- [S44] Kurtukova, Anna, Aleksandr Romanov, and Alexander Shelupanov. "Source Code Authorship Identification Using Deep Neural Networks." Symmetry 12.12 (2020): 2044.
- [S45] Meng, Xiaozhu. "Fine-grained binary code authorship identification." Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. 2016.
- [S46] Dutra, Eliezer, Bruna Diirr, and Gleison Santos. "Human factors and their influence on software development teams-a tertiary study." Brazilian Symposium on Software Engineering. 2021.
- [S47] Ju, An, et al. "A case study of onboarding in software teams: Tasks and strategies." 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021.
- [S48] Ktata, Oualid, and Ghislain Lévesque. "Designing and Implementing a Measurement Program for Scrum Teams: What do agile developers really need and want?." Proceedings of the Third C* Conference on Computer Science and Software Engineering. 2010.
- [S49] Sadowski, Caitlin, Kathryn T. Stolee, and Sebastian Elbaum. "How developers search for code: a case study." Proceedings of the 2015 10th joint meeting on foundations of software engineering. 2015.
- [S50] Lee, Michael J., and Andrew J. Ko. "Representations of user feedback in an Agile, collocated software team." 2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE). IEEE, 2012.
- [S51] Tufano, Michele, et al. "When and why your code starts to smell bad." 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 1. IEEE, 2015.
- [S52] Shastri, Yogeshwar, Rashina Hoda, and Robert Amor. "Understanding the roles of the manager in agile project management." Proceedings of the 10th Innovations in Software Engineering Conference. 2017.
- [S53] Cataldo, Marcelo, James D. Herbsleb, and Kathleen M. Carley. "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity." Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. 2008.
- [S54] Rahman, Foyzur, and Premkumar Devanbu. "How, and why, process metrics are better." 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.
- [S55] Siy, Harvey, Parvathi Chundi, and Mahadevan Subramaniam. "Summarizing devel-

- oper work history using time series segmentation: challenge report.” Proceedings of the 2008 international working conference on Mining software repositories. 2008.
- [S56] McLeod, Laurie, and Stephen G. MacDonell. ”Factors that affect software systems development project outcomes: A survey of research.” *ACM Computing Surveys (CSUR)* 43.4 (2011): 1-56.
- [S57] Abuhamad, Mohammed, et al. ”Large-scale and Robust Code Authorship Identification with Deep Feature Learning.” *ACM Transactions on Privacy and Security (TOPS)* 24.4 (2021): 1-35.
- [S58] Freire, Arthur, et al. ”A Bayesian networks-based approach to assess and improve the teamwork quality of agile teams.” *Information and Software Technology* 100 (2018): 119-132.
- [S59] Radhakrishnan, Abirami, et al. ”The impact of project team characteristics and client collaboration on project agility and project success: An empirical study.” *European Management Journal* (2021).
- [S60] Tenenberg, Josh. ”An institutional analysis of software teams.” *International Journal of Human-Computer Studies* 66.7 (2008): 484-494.
- [S61] Tessem, Bjørnar. ”Individual empowerment of agile and non-agile software developers in small teams.” *Information and software technology* 56.8 (2014): 873-889.
- [S62] Avelino, Guilherme, et al. ”Measuring and analyzing code authorship in 1+ 118 open source projects.” *Science of Computer Programming* 176 (2019): 14-32.
- [S63] Crowston, Kevin, et al. ”Self-organization of teams for free/libre open source software development.” *Information and software technology* 49.6 (2007): 564-575.
- [S64] Drury-Grogan, Meghann L., Kieran Conboy, and Tom Acton. ”Examining decision characteristics challenges for agile software development.” *Journal of Systems and Software* 131 (2017): 248-265.
- [S65] Nuñez-Varela, Alberto S., et al. ”Source code metrics: A systematic mapping study.” *Journal of Systems and Software* 128 (2017): 164-197.
- [S66] Shihab, Emad, et al. ”Is lines of code a good measure of effort in effort-aware models?.” *Information and Software Technology* 55.11 (2013): 1981-1993.
- [S67] Burga, Ruben, et al. ”Examining the transition to agile practices with information technology projects: Agile teams and their experience of accountability.” *International Journal of Project Management* (2021).
- [S68] Capiluppi, Andrea, Nemitari Ajenka, and Steve Counsell. ”The effect of multiple developers on structural attributes: a study based on java software.” *Journal of Systems and Software* 167 (2020): 110593.
- [S69] Shastri, Yogeshwar, Rashina Hoda, and Robert Amor. ”The role of the project manager in agile software development projects.” *Journal of Systems and Software* 173 (2021): 110871.
- [S70] Abuhamad, Mohammed, et al. ”Code authorship identification using convolutional neural networks.” *Future Generation Computer Systems* 95 (2019): 104-115.
- [S71] Trainer, Erik H., and David F. Redmiles. ”Bridging the gap between awareness and trust in globally distributed software teams.” *Journal of Systems and Software* 144 (2018): 328-341.
- [S72] Gote, Christoph, Ingo Scholtes, and Frank Schweitzer. ”Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net.” *Empirical software*

engineering 26.4 (2021): 1-41.

[S73] Bertram, Dane, et al. "Communication, collaboration, and bugs: the social nature of issue tracking in small, colocated teams." Proceedings of the 2010 ACM conference on Computer supported cooperative work. 2010.

[S74] Kakar, Adarsh Kumar Satindarlal. "Engendering cohesive software development teams: Should we focus on interdependence or autonomy?." International Journal of Human-Computer Studies 111 (2018): 1-11.

[S75] Hundhausen, Christopher, et al. "Evaluating Commit, Issue and Product Quality in Team Software Development Projects." Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. 2021.

[S76] Moedt, Wouter, et al. "Enhancing IoT Project Success through Agile Best Practices." ACM Transactions on Internet of Things (2022).

[S77] Zabardast, Ehsan, Javier Gonzalez-Huerta, and Binish Tanveer. "Ownership vs Contribution: Investigating the Alignment Between Ownership and Contribution." 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). IEEE, 2022.

[S78] Kantola, Konsta, Jari Vanhanen, and Jussi Tolvanen. "Mind the product owner: An action research project into agile release planning." Information and Software Technology 147 (2022): 106900.

[S79] Lupea, Mihaela, et al. "SoftId: An autoencoder-based one-class classification model for software authorship identification." Procedia Computer Science 207 (2022): 716-725.

Appendix (II) - Studies and Variables Explanation

Table 4.1: Specifications of the papers directly relevant to "Ownership" of software artifacts included in our systematic literature review. Paper which did not discuss ownership directly (such as S66) are excluded from this table.

| Paper What? | | Who? | How? | Dependent variable | Independent variable | Control variable |
|-------------|---------------------|-----------------|-------------------|---------------------------------|--|---|
| S1 | Code, Component | Developer | Collective shared | Number of coding errors | Expertise location, Expertise aware | Developer experience, team size, and project size |
| S2 | Product | Product manager | Collective shared | N/A | N/A | N/A |
| S3 | Code (binary files) | Developer | Weighted shared | Number of post release bugs | Code Churn, Code Complexity, Code dependencies, Code coverage, Number of pre release bugs, Number of developers, Number of ex-developers, Edit frequency, Depth of master ownership, % of Org. contributing to development, Level of org. code ownership, Overall org. Ownership, Org. intersection factor | N/A |
| S4 | Code, File | Developer | Weighted shared | N/A | Code churn, Number of commits | N/A |
| S5 | Code, File | developer | Collective shared | N/A | N/A | N/A |
| S6 | Code, Component | Developer | Weighted shared | Number of post release failures | Code churn, Code complexity, Test coverage, Binary dependencies, Number of developers | Developer geographical location |

| | | | | | | |
|-----|-------------------|--------------------|-----------------|---|--|--|
| S7 | Code, Component | Developer, Manager | Weighted shared | Number of pre release failures, Number of post release failures | Highest % of contribution a developer made to a software component, Number of major contributor, Number of minor contributor, Number of developers | Code size, Code churn, Code complexity |
| S8 | Code, file | Developer | Weighted shared | Number of line of code to fix defect | Developer experience, Collective ownership | Project size, Number of bugs linked with bug fixing revision, Number of developers |
| S9 | Issue | Developer | Weighted shared | Number of active issues | Number of developers | N/A |
| S10 | Code | Developer | Weighted shared | Collective ownership of a class | Linguistic Topics using LDA | N/A |
| S11 | Component | Organization | Weighted shared | Number of pre release failures, Number of post release failures | Number of commits | Code size, Code churn, Code complexity |
| S12 | Code (authorship) | Developer | Weighted shared | N/A | N/A | N/A |
| S13 | Code (authorship) | Developer | Dedicated | Number of bugs | Characters churn per line per commit, | N/A |

| | | | | | | |
|-----|----------------|-----------|-----------------|--|---|---|
| S14 | Task,Review | Developer | Weighted shared | The time between review creation and last updating | The level of involvement of a review's owner in the overall review process | Number of patches associated with a review,The median of the approval scores received by a reviewer, The degree centrality of review in a network based on review similarity, Number of reviewers, Number of comments on a review by developers who do not own it, Number of reviews owned by its owner for each review |
| S15 | Code,Component | Developer | Weighted shared | Number of bugs | Highest % of contribution a developer made to a software component,Number of major contributor,Number of minor contributor,Number of developers | Code size, Code complexity |

| | | | | | | | |
|-----|-----------------|----------------------------------|-------------------|----------------------|------------|--|-----|
| S16 | Code, Component | Developer, Manager, Organization | Collective shared | Number bugs | of | Highest % of contribution a developer made to a software component, Number of developers, Number of minor contributor, Number of minimal contributor, Number of manager, Average ownership value of files in a directory, % of commits of highest contributor in a directory, % of commits of lowest contributor in a directory, Average contributors in a directory, % of minor contributor in a directory, % of minimal contributor in a directory, % of major contributor in a directory, Average of minimal contributor in a directory, Average of minor contributor in a directory, Lowest ownership value of file , Number of files with 50% ownership value | N/A |
| S17 | Code | Developer | Weighted shared | Fractional ownership | | Number of change of every character in a source code file | N/A |
| S18 | Code,commit | Developer | Weighted shared | Number bugs | of | Number of developers,Number of commits | N/A |
| S19 | Code, Component | Developer | Weighted shared | Number post bugs | of release | Code size, Code churn, Number of developers, Number of files touched by a developer, Highest % of contribution a developer made to a software component, Number of major contributor, Number of minor contributor | N/A |
| S20 | Code | Developer | Weighted shared | N/A | | N/A | N/A |

| | | | | | | |
|-----|-----------------|---|-----------------|--|---|---|
| S21 | Code, component | Developer, Reviewer | Weighted shared | Number of post release defects | Highest code ownership value, Highest review specific ownership value, Proportion of minor developer and major reviewer, Proportion of major developer and major reviewer, Proportion of major developer and minor reviewer, Proportion of minor developer and minor reviewer, Proportion of minor developer and major reviewer | Code size, Code churn, Entropy, Number of developer, Number of reviewer |
| S22 | Code | Developer | Collective | N/A | N/A | N/A |
| S23 | Code, file | Developer | Weighted Shared | Number of faults | Number of major contributor, Number of minor contributor, Number of developers, Highest % of contribution a developer made to a software component | Project size, Code complexity |
| S24 | Bug fix | Developer | Dedicated | Code churn, Bug fixing time | Authorship of bug fixing code | N/A |
| S25 | Test cases | Developer (Test Engineer), organization | Weighted shared | Number of fixed bugs, Relative number of fixed bugs per test suite execution | Number of engineers and ex-engineers who contributed to a test suite, Length of organizational communication path, Number of mailing list and user account as test owner in a test suite | N/A |
| S26 | Project | Developer, Department | Weighted shared | Number of failures per hour | Number of developers, Number of new developers, Expansion acceleration, Department modularity | N/A |
| S27 | Code, commit | Developer | Dedicated | N/A | N/A | N/A |

| | | | | | | |
|-----|-----------------|-----------|---------------------------------|---|--|-------------------------------|
| S28 | Build | Developer | Dedicated, Collective shared | Build change type, Build ownership style | % of all build commits of a project that belongs to a specific category (Adaptive, Corrective, Perfective, Preventive, new functionality, Reflective) of build changes, Average amount of build churn per file changed by the commits of a build change category (Adaptive, Corrective, Perfective, Preventive, new functionality, Reflective), The median number of unique build files modified by the build commits in a build change category (Adaptive, Corrective, Perfective, Preventive, new functionality, Reflective) | N/A |
| S29 | Code,Commit | Developer | Weighted shared | Number of post-release defects | Number of developers, Number of major contributor, Number of minor contributor, Highest % of contribution a developer made to a software component | Code Size, Code complexity |
| S30 | Code, File | Developer | Weighted shared | Number of refactored files | Mean ownership computed on file,Mean ownership computed on author | N/A |
| S31 | Code | Developer | Shared | N/A | N/A | N/A |
| S32 | Product | Developer | Collective shared | Turnover Intention | Value Fit, Demands-Abilities Fit , Psychological ownership | Financial Compensation |
| S33 | Code | Developer | Collective Shared | N/A | N/A | N/A |
| S34 | Code, component | Developer | Weighted shared | N/A | N/A | N/A |

| | | | | | | |
|-----|-----------------------------|------------------------------|-------------------|-----|--|-----|
| S35 | Code, component (fragment) | Developer | Collective Shared | N/A | N/A | N/A |
| S36 | Code (authorship) | Developer | Weighted shared | N/A | Programming style and code smell metrics based on variables and natural language | N/A |
| S37 | Code (authorship) | Developer | Weighted shared | N/A | N/A | N/A |
| S38 | Requirement | Developer, User, Stakeholder | Shared | N/A | N/A | N/A |
| S39 | Code (authorship) | Developer | Dedicated | N/A | N/A | N/A |
| S40 | Code | Organization | Weighted shared | N/A | N/A | N/A |
| S41 | Code, fragment (authorship) | Developer | Shared | N/A | N/A | N/A |
| S42 | Code | Developer | Dedicated | N/A | N/A | N/A |
| S43 | Code, File, commit | Developer | Collective shared | N/A | Code size, Number of line of code of each author | N/A |
| S44 | Code (authorship) | Developer | Weighted shared | N/A | N/a | N/A |
| S45 | Code, Functions | Developer | Dedicated | N/A | N/A | N/A |
| S46 | Project | Developer, Team | Collective shared | N/A | N/A | N/A |
| S47 | Task | Developer | Weighted shared | N/A | N/A | N/A |

| | | | | | | |
|-----|-----------------------------|-----------|-------------------|---|--|--------------|
| S48 | Product | Developer | Dedicated | | | |
| S49 | Code | Developer | Dedicated | N/A | N/A | N/A |
| S50 | Issue | Developer | Shared | N/A | N/A | N/A |
| S51 | Code,File, Commit | Developer | Dedicated | N/A | N/A | N/A |
| S52 | Task | Developer | Weighted shared | N/A | N/A | N/A |
| S53 | Code, file | Developer | Dedicated | N/A | N/A | N/A |
| S54 | Code | Developer | Dedicated | Number of de- fects | Developer experience, Code size | N/A |
| S55 | File | Developer | Dedicated | N/A | N/A | N/A |
| S56 | Project | User | Shared | N/A | N/A | N/A |
| S57 | Code (source and binary) | Developer | Dedicated | N/A | N/A | N/A |
| S58 | Product | Manager | Dedicated | N/A | N/A | N/A |
| S59 | Product | Developer | Dedicated | N/A | N/A | N/A |
| S60 | Code | Developer | Shared | N/A | N/A | N/A |
| S61 | Code and Product | Developer | Collective Shared | N/A | N/A | N/A |
| S62 | Code, file | Developer | Shared | Degree of au- thorship of a specific devel- oper | First authorship of developer through file creation, Number of changes made by file creator developer, Number of changes made by other developer ex- cept file creator | Project size |
| S63 | Code | Developer | Shared | N/A | N/A | N/A |

| | | | | | | |
|-----|-------------------|--------------------|------------------------------|----------------------------|--|--|
| S64 | Task | Developer | Dedicated | N/A | N/A | N/A |
| S65 | Code | Developer | Shared | N/A | Code size | N/A |
| S67 | Task, Project | Developer, Manager | Dedicated | N/A | N/A | N/A |
| S68 | Code | Developer | Weighted shared | N/A | N/A | N/A |
| S69 | Product | Manager | Dedicated | N/A | N/A | N/A |
| S70 | Code (authorship) | Developer | Weighted shared | N/A | N/A | N/A |
| S71 | Code | Developer | Shared | N/A | N/A | N/A |
| S72 | Code, File | Developer | Weighted shared | N/A | N/A | Code complexity, Developer experience, Type of project |
| S73 | Issue | Developer | Dedicated | N/A | N/A | N/A |
| S74 | Code | Developer | Collective shared | N/A | N/A | N/A |
| S76 | Code | Developer | Collective shared | N/A | N/A | N/A |
| S77 | Code | Developer | Dedicated, Collective shared | Proportion of contribution | Number of commits, Code churn, Code complexity, Ticket complexity, Pull Request complexity, Number of pull requests, Number of Tickets | N/A |
| S78 | Product | Product owner | Collective shared | N/A | N/A | N/A |
| S79 | Code | Developer | Collective shared | N/A | N/A | N/A |