Automation in Open Source Software: A GitHub Marketplace Analysis

SK GOLAM SAROAR

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

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

YORK UNIVERSITY
TORONTO, ONTARIO

June 2023

© SK GOLAM SAROAR, 2023

Abstract

This thesis comprises two papers that examine automation tools in the Open Source Software (OSS) ecosystem on GitHub, focusing on GitHub Actions as well as the GitHub Marketplace, which is a platform for sharing these Actions for collaboration and reuse. Our research aims to understand and explore the state of automation in OSS, as existing studies have mainly focused on statistical analysis of a sample of GitHub repositories, neither considering developers' perspectives nor leveraging the GitHub Marketplace. The first paper conducted a survey analysis to investigate the motivations, decision criteria, and challenges associated with creating, publishing, and using Actions. The second paper explores the GitHub Marketplace and presents a mapping study by analyzing 7,878 Actions and 515 research papers mapped into 32 different categories. We found a substantial industry-academia gap, with researchers focusing on experimentation and practitioners relying more on exploration tools. The limited number of OSS automation tools published in academia contrasted with the convenient access practitioners had to the marketplace offerings. This thesis contributes to the understanding of automation in the OSS ecosystem, highlights the industry-academia gap, offers insights for researchers to build on existing work, and aids practitioners in navigating technology and finding synergies.

To my mother, with love . . . and a heavy heart for the eight thousand miles between us.

Acknowledgments

Completing this thesis would not have been possible without the support, guidance, and encouragement of many individuals, to whom I would like to express my deepest appreciation and gratitude.

Firstly, I would like to thank my supervisor, Dr. Maleknaz Nayebi, for her unwavering support and guidance throughout my graduate studies. Her expertise, patience, and encouragement have been invaluable, and I am grateful for her mentoring, which has been a critical factor in my academic and personal growth.

I would also like to express my gratitude to my supervisory committee members, Dr. Song Wang and Dr. Amaneh Jadidi, for their valuable feedback and insightful suggestions that helped improve the quality of my thesis.

I would like to extend my heartfelt appreciation to my lab mates for their constant support, collaboration, and friendship. Their enthusiasm and dedication to research have been a source of inspiration, and I have learned a lot from them.

I am especially grateful to Waseefa Ahmed, who made significant contributions to one of the papers in my thesis. Her hard work, knowledge, and expertise have been critical in completing that paper, and I am thankful for her dedication and commitment.

I am also thankful to the department for providing me with the necessary resources and facilities to conduct my research. Their support has been essential in helping me overcome various challenges during my graduate studies.

Finally, I would like to express my heartfelt gratitude to my family, friends, and

in particular, my beloved wife Fairouz, for their love, encouragement, and support throughout my academic journey. Their consistent backing, understanding, and patience have been a constant source of motivation, and I am grateful for their presence in my life.

Table of Contents

Abstract iii

Acknowledgments v

Table of Contents v

List of Tables viii

List of Figures ix

Chapter 1: Introduction 1

1.1 Automation on GitHub 2

1.1.1 GitHub Actions 3

1.1.2 GitHub Apps 5

1.2 Software Marketplaces 5

1.2.1 Mobile app stores 5

1.2.2 The GitHub Marketplace 8

1.2.3 Other Software Marketplaces 10

1.3 Thesis Overview 11

Chapter 2: Developers' Perception of GitHub Actions: A Survey Analysis 13

2.1 Introduction 14

2.2 Related Work 18

2.3 Methods 20

2.3.1 Survey Design . . . 21
2.3.2 Selecting participants . . . 24
2.3.3 Analyzing the data . . . 25
2.4 Results . . . 26
2.4.1 Survey Participants . . . 26
2.4.2 **RQ1** - Developers' motivation for creating, publishing, and reusing Actions . . . 27
2.4.3 **RQ2** - Decision criteria for developers to create and use Actions 31
2.4.4 **RQ3**- Challenges for developers in creating, publishing, and using Actions . . . 38
2.5 Discussion and Recommendations . . . 41
2.6 Threats to Validity . . . 44
2.7 Conclusion . . . 45

Chapter 3: GitHub Marketplace for Practitioners and Researchers to Date 47
3.1 Introduction . . . 48
3.2 Related Work . . . 52
3.2.1 Mining GitHub Workflows . . . 53
3.2.2 Mining Marketplaces in Software Engineering . . . 55
3.3 Research Methodology . . . 56
3.3.1 Process of Gathering Data from the Marketplace . . . 57
3.3.2 Protocols Used for Systematic Mapping Study . . . 58
3.3.3 Gap Analysis . . . 63
3.4 **RQ1**: Observations from Mining GitHub Marketplace . . . 65
3.4.1 Developers Attributes . . . 66
3.4.2 Platform Attributes . . . 74
3.4.3 User and Usage Attributes . . . 75

3.5 **RQ2**: Mapping Studies of Open Source Software Engineering Literature . . . 78
3.5.1 Mapping to the Types of Analytical Tools . . . 79
3.5.2 Mapping of Research Type and Replication Status . . . 81
3.5.3 Mapping into GitHub Categories . . . 83
3.6 RQ3: The Gap . . . 88
3.6.1 Comparison of Nature of the Analytical Tools . . . 88
3.6.2 Tool Availability . . . 91
3.6.3 The Popularity of Automation Tasks . . . 91
3.6.4 Overlap Between Categories in Literature and in the Marketplace . . . 96
3.6.5 Qualitative Comparison of Automation Tools in Research and Practice . . . 98
3.7 Discussion and Implications . . . 100
3.8 Threats to Validity . . . 103
3.9 Conclusion . . . 106

Chapter 4: Conclusion and Future Work **108**
4.1 What's Next? . . . 108
4.2 Wrapping Up . . . 110

Bibliography **112**

Appendix A: Categorized Papers **123**

Appendix B: List of the Mapped Studies **127**

List of Tables

2.1 Comparison between existing literature and our work 17
2.2 Survey questions for each RQ and their recipients 21
2.3 Motivations and challenges for creating Actions 28
2.4 Advantages of publishing Actions to the marketplace and reasons why developers do not publish. 29
2.5 How developers decide, find, and use Actions and the challenges they encounter 32
2.6 Debugging workflow (YAML) files and using GitHub Marketplace . 39

3.1 Publication venues selected from Google Scholar for exclusion Criteria V and the number of papers studied from each venue. 60
3.2 Categories of apps and actions on the GitHub Marketplace. 66
3.3 Top features in each marketplace category (from product descriptions) and their frequencies in parentheses. 71
3.4 Number of mapped papers using the schema of analytical tools [111]. 80
3.5 Number of papers in the mapping study using the research type [113] 81

A.1 Papers mapped to GitHub Marketplace categories 123

List of Figures

2.1 The stages of our survey method 20
2.2 Tasks developers automate using GitHub Actions. 31
2.3 Developer preferences for choosing an Action when there are multiple Actions to choose from. 35

3.1 Relation between the Research Questions in this study 51
3.2 Attributes on GitHub Marketplace categorized by apps, actions, or both. 57
3.3 The process of our systematic mapping with the number of papers retrieved at each step. 61
3.4 Number of apps and actions per category. 68
3.5 Frequency of categories appearing together for apps and actions on the marketplace. 69
3.6 Number of characters in short descriptions for apps and actions on the GitHub Marketplace 70
3.7 Number of developers (apps and actions) per category 75
3.8 Distribution of number of installs for apps and number of stars for actions on the marketplace 76
3.9 Number of categorized papers within our systematic mapping study (**RQ2**) in each year 79
3.10 Papers in our mapping study: Research type and type of analysis. . . 82

3.11 Replication status of papers in our systematic mapping study on automation tools in open source 83
3.12 Number of papers in each year per category. 84
3.13 Frequency of categories appearing together for the papers in our analysis 85
3.14 Number of authors per category 87
3.15 The number of marketplace products and the number of papers in the spectrum of analytic tools for development activities adopted from [111]. 89
3.16 Difference in the number of practitioners and the number of researchers, and the percentage of products and papers in each category. 92
3.17 Category rankings based on three factors: average paper citations per year, average app installs, and average action stars. 94
3.18 Intersection between categories in research and practice: largest overlaps between categories 97
3.19 Word clouds for the categories Localization and Utilities, based on app descriptions. 104

Abbreviations & Acronyms

CSP Cloud Service Provider.

ML Machine Learning.

OSS Open Source Software.

SLR Systematic Literature Review.

Chapter 1

Introduction

Automation of software development processes is one of the major contributors to developing better software products [1]. `Automation` refers to the process of using technology to perform tasks with reduced human assistance [2]. Multiple studies have shown that automation can improve efficiency, speed, and accuracy in software development [3], [4]. Companies that embrace automation can deliver high-quality software products at a faster pace and with fewer errors [5]. According to a systematic literature review (SLR) by Rafi et al. [6], automated software testing has been shown to have benefits such as increased development productivity through high test reusability, time-saving and cost reduction, better test coverage, and improved product quality. Additionally, in a study by Ståhl et al. [7], the authors found that automation techniques such as continuous integration resulted in improved developer productivity, software predictability and transparency, as well as faster build, integration, and release cycles. Another SLR by Soares et al. [8] reported that automation in software development can help to identify defects earlier in the development process, thereby reducing the cost of fixing defects. An SLR by Ng et al. [9] reported that intelligent automation offers sophisticated decision analysis, data-driven decision-making, and scenario analysis in engineering applications and automated business processes, eliminating operational errors and automating rule-based, well-structured, and repetitive decisions.

GitHub is a web-based platform that provides version control and collaboration features to support software development projects. GitHub is the world's largest open-source community [10], allowing users to create and manage repositories of source code, track changes made to code over time, and collaborate with others through features like pull requests and issues [11]. With millions of users and repositories, GitHub has become an essential tool for developers working on both open-source and enterprise software projects. GitHub introduced its marketplace to gather and share automation tools for GitHub-hosted projects. Third-party developers provide tools on GitHub Marketplace to automate workflows in software repositories. The majority of these tools are open source. Open-source software refers to software that is developed and maintained by open collaboration and made freely available for use, review, modification, and redistribution by anyone [12]. We will discuss the GitHub Marketplace in section 1.2.2 before exploring it in greater depth in Chapter 3. But first, we will have a look at the automation tools on GitHub.

1.1 Automation on GitHub

Automation tools or bots can be integrated into GitHub repositories to streamline and automate development workflows. These bots can perform a range of tasks, such as automating code reviews, labeling issues, and updating pull requests [13]. By using GitHub Bots, development teams can improve their productivity, reduce manual work, and enhance the overall quality of their codebase [14].

Many projects use bots to automate a wide range of tasks such as refactoring source code [15], fixing bugs [16], updating project dependencies [17], supporting communication and decision-making [18], etc. Wu et al. [14] performed a preliminary study of the use of bots in the GitHub community and found that 56% of the sampled projects had used bots extensively, and most projects had used more

than one bot. The researchers categorized bots according to their usage in projects and reported that 40% of projects used bots for project management, followed by code management (28%), and testing and deployment (17%). Wessel et. al [19] investigated the usage and impact of such bots and asked developers about their desired improvements for bots. Developers wanted smarter bots that would decrease code review effort, decrease the time to close pull requests, and automate continuous integration tasks. To assist students in checking code style and functioning, Hu et al. [20] built up a static code analyzer and a continuous integration service on GitHub. The authors implemented three bots and showed that more than 70% of students believed the bots' advice was valuable and that they could deliver much more feedback than teaching professionals.

The study by Storey et al. [18] showed that while the impact of bots on development workflows may be limited, developers generally recognize the potential of bots and desire intelligent automation tools. Ultimately, the use of GitHub Bots has played a significant role in automating the software development process on GitHub. GitHub recently introduced Actions and Apps, each offering unique capabilities to enhance the software development workflows on GitHub.

1.1.1 GitHub Actions

While GitHub Bots offer a great way to automate various tasks in development workflows, they do have some limitations. One significant limitation is that bots have less granularity in permission scope and provide limited control over the API calls they can make [21], [22]. Additionally, implementing authentication for these bots can be complex and require additional development effort, especially for bots that require a high level of security [23].

To address some of these limitations, GitHub offers GitHub Actions, which provides a more powerful and flexible way to automate workflows. GitHub Actions

can be used to automate tasks like building, testing, and deploying applications directly from GitHub repositories. Actions workflows are triggered when an event occurs in a GitHub repository. For example, a developer wants to automatically run their test suite every time they push code to their repository. With GitHub Actions, they can write a workflow that specifies the steps to run the tests and trigger the workflow to run on every `push` event. This means that every time a developer pushes code, GitHub Actions will automatically build and run the tests, giving the developer immediate feedback on whether their changes introduced any errors. A repository can contain several workflows, each with its own specific tasks. For instance, we may have a workflow designed to build and verify pull requests, another to deploy our application whenever a new release is created, and yet another that adds a label each time a new issue is created. GitHub Actions can be easily customized using a YAML file and can be shared and reused across different repositories. In fact, developers have the flexibility to choose from a wide range of Actions for their particular automation task. This enables them to switch to alternative Actions that offer better quality, additional features, or improved compatibility. In other words, when faced with limitations such as bugs or inadequate documentation in their current Action, developers can opt for a different Action that better meets their requirements (see Table 2.5 in Chapter 2).

Despite the abundance and numerous benefits of GitHub Actions, developers choose *not* to use Actions in certain scenarios. These decisions are primarily driven by factors such as the perceived cost/effort trade-off in using Actions, increased project complexity, potential security risks, inadequate documentation, or a personal preference for manual task execution. A detailed overview of these considerations can be found in Table 2.5 in Chapter 2 of the thesis.

1.1.2 GitHub Apps

GitHub also introduced GitHub Apps, which are more versatile and secure than bots [23]. GitHub Apps can perform authenticated API requests and can also receive and respond to webhook events. This means that GitHub Apps can perform more complex tasks, such as automatically merging pull requests [24], managing project boards [25], and even monitoring Actions workflows [26]. There are cases when Apps should be used rather than Actions, for example, when accessing the team's workflow requires further permissions to the repository, or when the repository owner would need to host and run the automation, or when there is a need for a solid architecture of the hosted data. However, GitHub Apps require users to install them on a specific repository or organization, which can limit their scope of operation.

1.2 Software Marketplaces

A software marketplace is a digital platform that serves as an intermediary between a software supplier organization and its customers [27]. These marketplaces offer a centralized location for developers to find and download tools and frameworks for their use cases [28]. Previous research has highlighted the growing popularity of platform-mediated software markets and the need to explore whether and how deployment portals for platform-specific software impact software development practices [29].

1.2.1 Mobile app stores

One of the most notable examples of a software marketplace is the mobile app store. Mobile app stores, such as Apple App Store, and Google Play, provide a useful platform for software developers to distribute and monetize their applications to a global audience. They offer a centralized location where users can eas-

ily discover and download new apps, as well as provide a system for developers to receive feedback and ratings from their users [30]. This centralized distribution system and the abundance of user data available within app stores also make them a valuable resource for researchers and analysts looking to gain insights into software development and user behavior. The concept of platform-mediated software engineering on two-sided markets, such as mobile app stores, has created a closed feedback loop with users and increased transparency in the market's software product status [29].

In their systematic literature review (SLR), Genc-Nayebi et al. [31] explored the domain of mining online opinions from app store user reviews and addressed the importance of app stores and user reviews in the mobile app ecosystem, the rapid growth of app store data mining studies, and the need for academic research in this field. The objectives of their SLR were to identify solutions for mining opinions, examine challenges, analyze contributions to software requirements evolution, and identify future research directions. Harman et al. [32] provided an insight into the importance of the marketplace ecosystem for software engineering practices by investigating the mobile app stores. Their findings suggest that app store mining can provide valuable insights for app developers, for example, they showed a strong correlation between customer rating and app downloads. Martin et al. [33] provided a survey of the research conducted in App Store Analysis, which combines non-technical information from app stores with technical information to analyze trends and behaviors in software repositories. Their study describes the impact of App Store Analysis on software development teams and highlights areas of research that have been explored, along with open problems and challenges that need to be addressed in future research. Jansen et al. [34] defined app stores and provided a conceptual model and typical features and policies observed in them, using six case studies. The aim was to help practitioners make their app store more

successful and provide researchers with a framework for defining and analyzing app stores. Mojica et al. [35] presented a study where they examined whether an app rating accurately reflects how the users perceive it. Their findings indicated that the app store's current metric fails to capture the users' changing levels of satisfaction as the app evolves. Ali et al. [36] conducted a large-scale comparative study of cross-platform apps collected from the Apple and Google Play app stores. The goal was to understand the differences in how users perceived the same app distributed through different platforms. Nayebi et al. [37] discussed the issue of mobile app releases not making it to the app store and introduced the concept of "marketability" as a new release decision problem. The authors analogical reasoning to guide the transition of not successfully marketable releases into successful ones and showed that predictive analytics and analogical reasoning can support marketability decisions for mobile apps. Comparing app-store attributes, such as stars, versions, and prices, and measuring the aggregated user-perceived ratings, they found many differences across the platforms.

Several other systematic literature reviews have been conducted on mobile app stores, exploring various fields beyond software engineering. For instance, one study by Wiemker et al. [38] aimed to quantify alcohol consumption using digital assessment tools available in app stores. Chen et al. [39] conducted research to provide valuable insights into the strengths and weaknesses of existing mobile apps in the context of adolescent and young adult pregnancy prevention. Ciapponi et al. [40] revealed a significant evidence gap regarding the accuracy of mobile apps in detecting substandard and counterfeit medicines. Additionally, several studies have examined the availability of resources for mental health within mobile app stores [41]–[44]. Others have explored the usage of mobile apps in health systems [45], studying triage apps for health emergencies [46], apps for calculating insulin dose [47], or apps developed to identify, treat, and prevent ulcers in adults

[48]. This is while the app store data may not provide a full spectrum of opinion for software engineering decision-making.

1.2.2 The GitHub Marketplace

The `GitHub Marketplace` is yet another platform-mediated software marketplace. It allows developers to provide free or paid tools to GitHub users for improving their workflows[1], in particular, when hosted on a GitHub repository. GitHub Marketplace offers both types of tools we discussed in the previous section- Apps and Actions. Third-party developers provide these automation tools on the marketplace to reduce the number of repetitive tasks within a workflow and increase the productivity of a software team. By using GitHub Marketplace, developers can easily find and integrate with the tools they need to improve their productivity and enhance the quality of their code-base. The benefits of the marketplace include a wide range of tools and integration, ease of use, and seamless integration with GitHub.

It is evident from the thousands of automation tools on the GitHub Marketplace that practitioners extensively use automation tools in their workflows. Moreover, the software engineering research community has devoted significant attention to automating development workflows and providing decision support to enhance the productivity of software teams and the value of their products. While researchers have studied other marketplaces, the GitHub Marketplace has never been the subject of an investigation to evaluate the alignment or difference between research efforts (the published studies) and the practice (automation tools on GitHub). In chapter 3, we will do just that. But before that, we will discuss how to publish tools on the marketplace, making them open-source in the process.

Different steps are required for adding apps and actions to the GitHub Market-

[1] https://docs.GitHub.com/en/developers/GitHub-marketplace/GitHub-marketplace-overview/about-GitHub-marketplace

place. To be successfully listed on the marketplace, apps must meet user experience requirements, branding requirements, in addition to billing requirements for paid apps. The *user experience requirements* state that apps must not sway users away from GitHub, must integrate with the platform beyond authentication, and must be publicly available on the marketplace. Also, listings must include valid contact information for the publisher, who should set up webhook events to get notified about GitHub plan changes or cancellations through the Marketplace API. Moreover, apps must have a relevant description and must specify a pricing plan. According to the *branding requirements*, apps must have a logo, feature card, screenshot images, and a well written description which are aligned with the recommendations provided by GitHub. Paid apps also have additional requirements. To publish a paid plan for an app (or an app that offers at least one paid plan) on the GitHub Marketplace, the app must be owned by an organization and the app developer must have owner permissions in the organization. Furthermore, the organization must be verified. At least 100 installations are required for GitHub apps to be listed as a paid app. Under the *billing requirements*, all paid apps need to handle GitHub Marketplace purchase events to manage new purchases, upgrades, downgrades, cancellations, and free trials.

To publish an Action, a developer first needs to create the Action in their repository. They need to ensure that the repository only includes the metadata file, code, and files necessary for the Action. This allows the developer to tag, release, and package the code in a single unit. The Action must be in a public repository and the repository must contain a single Action. The Action's metadata file should have a unique name and be located in the root directory of the repository. The name cannot match an existing GitHub Marketplace category as GitHub reserves the names of GitHub features. If these requirements are met, the developer can add the Action to the GitHub Marketplace by tagging it as a new release and publishing it

immediately on the GitHub Marketplace. Actions with the ✓icon, which is the *verified creator* badge, indicate that GitHub has verified the creator of the Action as a partner organization. Partners can email partnerships@github.com to request the verified creator badge[2]. However, verification is not a requirement for creating Actions, and may require additional time and resources from developers. Besides, while some developers may be part of larger organizations or projects that prioritize verification for security or trust purposes, most others may be working on small-scale or personal projects where verification is not a priority. For these reasons, not all developers go through this verification process. Only 2.66% of Actions have a verified creator on the marketplace [49].

1.2.3 Other Software Marketplaces

Another software marketplace that has gained popularity among developers in recent years is Docker Hub. This platform provides developers with a means to create and distribute containerized applications. Additionally, Docker Hub supports integration with popular CI/CD tools such as Jenkins and GitLab. Several studies have investigated the Docker Hub to analyze its security vulnerabilities [50]–[52], adoption of best practices [52], and potential issues [52], [53].

Cloud services are becoming more popular, and to make it easier for people to access these services, many cloud service providers have created their own marketplaces. These marketplaces act as a distribution channel, making it more efficient and transparent to support and distribute software services. Jung et al. [54] examined the security concerns of cloud users and the importance of having management policies in place. Their research aims to help users establish consistent criteria and requirements when selecting security services from a cloud marketplace. Park et al. [55] discussed the benefits and constraints of the cloud mar-

[2]https://docs.github.com/en/actions/creating-actions/publishing-actions-in-github-marketplace#about-badges-in-github-marketplace

ketplace for software distribution and deployment via a case study involving five global and local cloud marketplaces including the AWS Marketplace, and the ORACLE Cloud Marketplace. The cloud marketplace guarantees rapid cloud service construction and operation, contributing to overall cloud industry development. However, there are constraints such as the requirement for modification and API development in the form required by the CSP, the absence of infrastructure choice, and the cost of migration when changing cloud services.

The cloud allows for a wider adoption and advanced functionality of modern applications, which is facilitated by the creation of a marketplace. Marketplaces make it easier to develop applications through the reuse and aggregation of services and resources by enabling more products to participate in the service delivery process [56].

1.3 Thesis Overview

This thesis is based on papers either submitted or published. It consists of three chapters following the introduction.

In Chapter 2, we present our paper titled "Developers' Perception of GitHub Actions: A Survey Analysis". This paper is published at the International Conference on Evaluation and Assessment in Software Engineering (EASE '23). Previous mining studies in software engineering have investigated the adoption and usage of GitHub Actions for specific automation tasks such as continuous integration. However, the topic of developers' perception of GitHub Actions has received limited research attention. We performed a survey study with 90 software developers, who were recruited through personalized emails to GitHub developers (random sampling) and social media advertisements (convenience sampling) [57]. The aim of the research is to understand developers' perceptions and motivations for publishing and using GitHub Actions, as well as the challenges they face.

In Chapter 3, We present our second paper titled "GitHub Marketplace for Practitioners and Researchers to Date". The objectives of this paper are to conduct a comprehensive analysis of the GitHub Marketplace, providing insights into the tools and automation practices used in the GitHub ecosystem, and to compare these findings with existing research in OSS automation, highlighting areas of alignment and potential gaps between industry and academia. We scraped the GitHub Marketplace to collect information about 440 Apps and 7,778 Actions. We gathered another dataset comprising 515 research papers published between 2000 and 2021 in OSS automation, and manually classified them based on the predefined automation categories on GitHub Marketplace. Then we compared these research studies in software engineering with the automation tools in the marketplace in terms of automation categories, research type, analytical tools, and reproducibility. We highlighted the gaps between research and practice in open-source software automation.

In Chapter 4, we discuss the potential for future research and conclude the thesis by summarizing our contribution. In this thesis, we present the first study introducing and exploring GitHub Marketplace as a software repository. We also present the first study on the developers' perception of GitHub Actions. We provide a comprehensive analysis of both the GitHub Marketplace and GitHub Actions. This thesis can be valuable in the context of generative AI by providing insights into the practical needs and requirements of developers and researchers, which can facilitate the development and improvement of generative AI models and tools to better meet the automation needs of developers in the OSS ecosystem. Through this research, we shed light on the industry-academia gap in OSS automation. Furthermore, we identify and address practical challenges faced by developers, offering recommendations to enhance their workflows and maximize productivity.

Chapter 2

Developers' Perception of GitHub Actions: A Survey Analysis

Abstract

GitHub Actions is a powerful tool for automating workflows on GitHub repositories, with thousands of Actions currently available on the GitHub Marketplace. So far, the research community has conducted mining studies on Actions, with much of the focus on CI/CD. However, the motivation and best practices of developers for using, developing, and debugging Actions are unknown. To address this gap, we conducted a survey study with 90 Action users and developers. Our findings indicate that developers prefer Actions with verified creators and more stars when choosing between similar Actions, and often switch to alternative Actions when faced with bugs or a lack of documentation. We also found that developers find the composition of YAML files, which are essential for Action integration, challenging and error-prone. They primarily rely on Q&A forums to fix issues with these YAML files. Finally, we observed that developers would not likely adopt Actions when there are concerns around complexity and security risks. Our study summarizes developers' perceptions, decision-making process, and challenges in using, developing, and debugging Actions. We provide recommendations for improving the visibility, re-usability, documentation, and support surrounding Actions.

2.1 Introduction

GitHub, the most prominent social coding platform, provides features such as issue tracking and pull requests to facilitate open-source software development. This results in an increased workload for developers and the subsequent use of automation tools to perform various repetitive tasks. To increase workflow velocity, GitHub introduced GitHub Actions which brings automation directly into the software development lifecycle on GitHub. GitHub Actions (sometimes we refer to these as Actions) are built around GitHub events and are triggered by developers' activities, such as creating a pull request or opening a new branch on a repository [58].

Actions are the main leverage in the hand of software developers to automate their tasks on GitHub. The number of Actions in the GitHub Marketplace[1] increased by approximately 112% between April 2021 and December 2022, with the number of Actions rising from 7,878 to 16,730 during this time period [49]. Even further, not all actions are distributed on the GitHub marketplace, and many developers design and keep their Actions within local repositories and never put them on the marketplace [59]. Decan et al. [60] analyzed the most frequent automation practices on GitHub and found that 43.9% of repositories in their dataset adopt and use Actions for automating parts of their workflows.

In this environment, for any given task, multiple competing Actions can be expected. This raises a new challenge for the practitioners to decide which Action to use [59]. GitHub Actions are created *by* the developers *for* the developers. These Actions reflect a real-world effort for task automation in software teams. It is unclear how developers look up these Actions or what motivates them to choose a particular Action over the others. In addition, it is important to study "why" developers create new Actions despite having multiple alternatives to adopt or

[1] https://github.com/marketplace?type=actions

reuse. Further, to our knowledge, no research distinguishes developers who create and contribute to Actions (providers) from developers who only use Actions (users). We are the first to identify these different types of Action adopters to understand their motivation for developing and using (or not using) Actions and the challenges associated with these tasks.

So far, the software engineering mining community investigated the usage and adoption of GitHub Actions but not Action development. A number of mining studies evaluate the basic adoption and usage details of GitHub Actions. These studies retrieve a random set of repositories that use and adopt Actions and study a particular automation task. For instance, Golzadeh et al. [61] focused on continuous integration (CI) and mined 91,810 GitHub repositories to analyze how the CI landscape has changed since GitHub Action was introduced. They reported that the introduction of Actions aligns with a decreasing growth rate of CI usage for Travis, CircleCI, and Azure. Kinsman et al. [59] analyzed 3,190 repositories to investigate how several development activity indicators change after Action adoption. In our study, to understand developers' perceptions, we conducted a survey study involving a total of 90 software developers on GitHub. The goals of our study are to understand (i) why and how developers publish or use GitHub Actions, and (ii) what challenges developers face in creating, publishing, or using Actions. Through an exploratory analysis [62] of the survey responses, we answer the following research questions:

RQ1- What motivates software developers to create, publish, and use GitHub Actions?

What and How: We are interested to understand why developers create Actions, whether or not they study existing Actions before creating their own, why they publish Actions on the marketplace, and which tasks they automate using Actions. Therefore, we surveyed participants with two close-ended and two open-ended

questions.

RQ2- What are the decision criteria for developers to create and use GitHub Actions?

What and How: After asking developers why they create and use Actions, we move on to explore how they do it. We surveyed participants to learn when they automate tasks (and when they do not), how they find Actions, choose between similar Actions, and configure and debug issues with their workflow files. To this end, we had a total of eight questions, one of which was open-ended.

RQ3- What challenges do developers face in creating, publishing, and using Actions?

What and How: We analyzed the challenges in creating Actions, publishing them to the marketplace, as well as adopting and configuring the workflow files when using the Actions. We also asked participants about the issues in workflow automation and the GitHub marketplace in general. We asked nine questions, five of which were open-ended.

This study is the first to distinguish different kinds of Action adopters and try to understand their perceptions of GitHub Actions. We designed a survey and collected data involving the responses of 90 software engineers. We provide a detailed analysis of the survey responses that give insights on Action providers and users-their motivation, decision criteria, and challenges in creating, publishing, and using Actions. Our findings can be used to help users choose the right Actions for their development tasks and help providers identify areas of improvement. This should also help researchers understand the state of practice in software automation so that they can conduct relevant research to advance the community.

Section 2.2 presents the related work and Section 2.3 describes survey design, participant selection, and data analysis. Section 2.4 presents the results of the survey analysis. Section 2.5 discusses the implications and recommendations. Section

Table 2.1: Comparison between existing literature and our work

Existing study	**Contributions**	**Comparison**
Kinsman et al. [59], Wessel et al. [63]	Mined GitHub repositories to understand the impact of Actions on the pull-request process. To understand developers' perceptions of Actions, they analyzed comments on GitHub issues that discuss Actions.	We focus on a qualitative study and survey developers to better understand their perceptions of Actions. For example, while some developers discuss issues in the Action repository, our survey data shows that most developers use Q/A forums such as Stack Overflow to discuss issues regarding Actions.
Chen et al. [64]	Also mined GitHub repositories to understand the impact of Actions on the commit frequency, pull request, and issue resolution efficiency. To investigate how developers configure their Actions, they performed an analysis of the YAML files used in Action repositories.	While a statistical analysis such as [64] gives a useful overview of developer practices in writing YAML files, it is important to understand how they write these files, what issues they face, and how they debug these issues. In our study, we survey practitioners on these topics and present our findings.
Decan et al. [60]	Analyzed the characteristics of the repositories that use GitHub Actions to study the automation practices, which workflows are automated, and which actions are reused.	Instead of studying repositories, we study developers to understand their motivation, decision criteria, and challenges in developing and using Actions. We ask developers when they automate a task, or why they do not reuse Actions.
Golzadeh et al. [61]	Mined GitHub repositories to analyze how the CI landscape has changed since Actions was introduced.	Although we asked developers what other automation services they use, we did not ask them if they migrated to Actions from other CI services.
Valenzuela-Toledo et al. [65]	Mined commits involving YAML files to highlight deficiencies in the way Actions workflows are produced and maintained.	We asked practitioners about their perceived challenges in writing YAML.

2.6 addresses the threats to validity and Section 2.7 concludes this paper.

2.2 Related Work

Kinsman et al. [59], [63] quantitatively analyzed the impact of adopting GitHub Actions, particularly focusing on how the dynamics of pull requests changed for 3,190 GitHub projects after they adopted Actions. Their results indicate that the adoption of Actions increases the number of rejected pull requests and decreases the number of commits in merged pull requests. In addition, the authors stated that adopting Actions led to requiring more time to accept a pull request. They also studied how the communities discuss GitHub Actions and reported that developers frequently ask for help with GitHub Actions by creating issues on GitHub. By manually inspecting 209 issues related to Actions, they concluded that developers have a positive perception of GitHub Actions. Similarly in [64], Chen et al. investigated the impact of Actions on the commit frequency, pull request, and issue resolution efficiency. They found that after adopting Actions, commit frequency decreases, the number of closed issues increases, and the number of pull requests decreases. However, unlike [63], the authors found that adopting Actions reduced the pull-request resolution latency. They also studied how developers configure Actions and opined that a template generation framework might be useful to ease the burden of GitHub Actions adoption. Decan et al. [60] analyzed the characteristics of the repositories that used GitHub Actions, which workflows were automated using GitHub Actions, as well as the most frequent automation practices on GitHub. They also studied which Actions were reused and found that most of the reused Actions were distributed by GitHub and belonged to the 'Utilities' or 'Continuous integration' categories. Golzadeh et al. [61] focused on continuous integration (CI) and analyzed how the CI landscape has changed since GitHub Action was introduced. They found that the introduction of Actions aligns with a decreasing growth rate of CI usage for Travis, CircleCI, and Azure. As the authors studied how frequently CIs are being replaced by an alternative, they found that

GitHub Actions attracted most of the migrations and took only 18 Months to overtake Travis CI in popularity. Even so, while there have been numerous research on the use [66]–[68], misuse [69]–[71], and areas of improvement [72], [73] for continuous integration, there is a substantial lack of similar research on GitHub Actions. Valenzuela-Toledo et al. [65] investigated the types of GitHub Actions workflows modifications performed by developers in 10 Popular GitHub repositories. They manually inspected 222 Commits related to workflow changes and identified 11 Different types of workflow modifications. They also presented opportunities for improvement in the way workflows are built and edited. The authors highlighted the need for adequate tooling to support refactoring, debugging, and code editing of Actions workflows. Calefato et al. [74] focused on GitHub Actions and CML[2] to present a preliminary analysis of the MLOps practices in GitHub. They studied 397 Actions workflows and 38 CML workflows to report whether workflow automation was prevalent on GitHub, what events were used to trigger these workflows, and what tasks were most frequently automated. The authors concluded that there were very few open-source projects involving ML-enabled components or systems that used GitHub Actions. Benedetti et al. [75] investigated the security issues affecting Action workflows and implemented a methodology (`GHAST`) for automatically assessing the presence of security issues in workflows. `GHAST` was used against 50 open-source projects to provide an overview of the current security landscape of Action workflows. Koishybayev et al. [76] created the workflow auditing GitHub Action, `GWChecker`, to help mitigate common security errors in the YAML configuration for CI/CD workflows. The workflow files are audited by `GWChecker` for plaintext secrets using regular expressions, versioning tags, non-verified Actions or Actions that are not available on the marketplace, and insecure triggers.

[2]https://cml.dev/

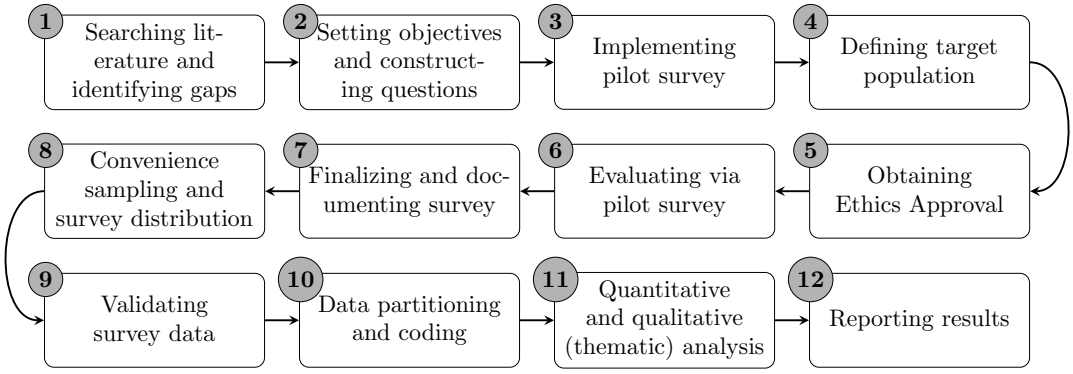


Figure 2.1: The stages of our survey method

Table 2.1 shows an overview of the prior studies related to Actions and how our work compares with them. All the studies rely on mining GitHub repositories and performing quantitative analysis for retrieving conclusions. To the best of our knowledge, this is the first study seeking developers' perceptions of the benefits and drawbacks of creating. publishing, and using GitHub Actions.

2.3 Methods

Following the empirical guidelines for survey design in software engineering [77], we discuss our survey design, participant selection, and analysis of the survey responses. The survey is fully reviewed and approved by the ethics board at York University (Certificate #: e2022-421). We followed the established empirical protocols for performing surveys following Kitchenham and Pfleeger [78]. In consideration of the current status quo of the state of the art and the nature of the mining studies on the repositories, we decided to focus this survey research on the *diagnostic evaluation* [62] to understand *"why"* and *"how"* of developers decision in creating and using GitHub Actions. Figure 2.1 shows the steps in our survey method.

2.3.1 Survey Design

We started by setting the objectives of the survey and performing a literature study. The objectives define the survey context and identify what areas to cover and what types of information are needed [77]. We identified three objectives for this survey to address the gaps in the mining studies and to better understand developers;

1. understanding *why* developers create, publish, and use Actions;
2. finding out *when* and *how* developers create, search for, switch between, and configure Actions,
3. identifying the *challenges* developers face in creating and using Actions as well as using the GitHub Marketplace.

Table 2.2: Questions in the survey and their recipients (P = Provider, U = User, NU = Non-user). Subscript with a question number shows the number of responses.

RQ1	**Rcpt**	**What motivates developers to create, publish, and use GitHub Actions?**
1_{44}	P	Can you specify why you did not reuse an existing Action but created one yourself? *(four options and others)*
2_{44}	P	When there are similar Actions (as yours) created by others, to what extent do you look into their functionality and scope? *(Likert scale)*
3_{44}	P	In your opinion, what are the advantages of publishing to GitHub Marketplace (if any)? *(textbox)*
4_{69}	P, U	What is/are the primary task(s) you are using GitHub Actions for? (mention up to three tasks) *(textbox)*
RQ2		**What are the decision criteria for developers to create and use GitHub Actions?**
5_{69}	P, U	How do you identify if a task should be automated in your GitHub workflow? (select all that apply) *(three options and others)*

Continued on next page

Table 2.2 – continued from previous page

6_{69}	P, U	When having a task requiring automation, what is your primary way of identifying a proper Action for use? (select all that apply) *(five options and others)*
7_{69}	P, U	When multiple Actions are available for a task, you choose the Action having __? (please drag the options to order them from highest to lowest preference according to you) *(six options and others)*
8_{69}	P, U	In the past, have you switched to using a different Action for your tasks? If yes, what was the reason for switching? (select all that apply) *(seven options and others)*
9_{69}	P, U	How do you write your workflow (YAML) files? (select all that apply) *(four options and others)*
10_{44}	P	Are you using self-hosted runners? *(yes/no)*
11_{44}	P	Why are you using self-hosted runners? *(five options and others)*
12_{69}	P, U	Have you chosen not to automate certain tasks even though GitHub Actions and other tools are available? If yes, please briefly specify which tasks and why. *(textbox)*
RQ3		**What challenges do developers face in creating, publishing, and using Actions?**
13_{44}	P	What challenges do you face in developing your Action(s) on GitHub (if any)? *(textbox)*
14_{44}	P	Have you published your Action on the GitHub Marketplace? If not, please specify why. *(four options and others)*
15_{69}	P, U	What challenges do you face in configuring your workflow (YAML) files (if any)? *(textbox)*
16_{25}	U	If facing an issue in configuring your YAML file, where do you primarily look for information? (select all that apply) *(four options and others)*
17_{69}	P, U	What challenges have you experienced in automating workflows using GitHub Actions (i.e. are there certain tasks that you could not automate)? *(textbox)*
18_{25}	U	GitHub Marketplace helps in discovering tools to improve workflows. Have you faced any challanges in using the marketplace? *(textbox)*
		Why do developers not use GitHub Actions?
19_{21}	NU	Have you heard about GitHub Actions for automating workflows in GitHub? *(yes/no)*

Continued on next page

Table 2.2 – continued from previous page

20_{21}	NU	Why do you not use GitHub Actions for automating your workflows? *(textbox)*
21_{46}	U, NU	Which other workflow automation services have you used for your GitHub repositories? (select all that apply) *(seven options and others)*
22_{25}	U	Do you believe that GitHub Actions have helped you in your role as a software engineer? *(5-point Likert scale)*

Having the objectives, we brainstormed on the potential questions for each objective. We filtered our questions based on similarity/redundancy and finalized 22 unique questions that were aligned with our objectives. We performed a second round of literature review to identify any relevant research and ensure that the questions are novel. We identified three types of developers based on their familiarity with GitHub Actions. Some developers created and published **(P)** Actions, and are highly familiar with the design, adoption, and maintenance decisions. However, not all creators are publishers as they might only use it within a limited context of their own repository. We will refer to this group as Providers. The second group has mostly adopted and used Actions to automate tasks in their repository, we refer to them as Users **(U)**. The third group, Non-users **(NU)**, are the least engaged users as they never used, and in some cases, never heard of GitHub Actions and hence not helpful in achieving our survey goals. To answer our research questions effectively, we curated our questionnaire for each of these groups. In Table 2.2, we list all the questions of the survey separated per objective and type of developers group (P, U, and NU).

Overall, the survey included three demographic questions, nine multiple-choice, two yes/no questions, two Likert-scale questions, one ranking question, and eight open-ended questions. The demographic questions concern the participants' experience in software development (i.e, less than 1 year, more than 10 years, etc.), their role in their company (i.e, software developer, project manager, other), and

their self-perceived level of familiarity with GitHub Actions (i.e created Actions, only used Actions, never used Actions, etc). To curate the survey based on the participants' familiarity and group type, for example, not asking developers what challenges they face in developing Actions while they have never developed or even heard of Actions, we used our third demographic question to design our survey in a way that participants were asked only the questions that applied to them. To build the survey instrument, we used `Qualtrics`. Our survey was anonymous, and we did not collect any identifying information about the participants.

2.3.2 Selecting participants

We invited 10 Developers from our close connections for a pilot study and evaluate our survey, i.e. making sure the questions were understandable, and the survey was *reliable* and *valid* [79]. A survey is reliable if we administer it many times and get roughly the same distribution of results each time. We employed the *alternative form reliability* where we asked the invitees to participate twice but reworded a few questions the second time. On the other hand, *validity* is concerned with how well a survey measures its' objectives. We used *content validity* where we asked invitees to provide us with written feedback (if any) regarding the survey. We also asked them to record how many minutes it took them to complete the survey. We analyzed the pilot responses to assess if we needed to change anything about the survey before we sent it out to a larger pool of participants. Based on the feedback from our initial participants, we revised two questions to eliminate ambiguity and set 15 minutes as the expected time to complete the survey. We recruited participants for the survey by sending personalized emails to GitHub developers (random sampling), and by advertising the survey on our social media (convenience sampling) [57]. We randomly sampled 250 developers who created or contributed to developing Actions and used our academic email to send personalized email invitations

In our survey invitation, we were careful not to spam the developers and sent emails to them just once. We created a personalized email for each user by (i) addressing the developer by his name in the email, and (ii) including the name of their Action in the email. This was done to cause the cognitive bias *halo effect* that would convince the participant to respond to our survey [80]. Authority and credibility also play a big part in getting invitees to respond to a survey [81]. To ensure that the developers considered our survey authentic and credible, we highlighted the nature of the research in both the email and the survey consent form. We also cited that the survey was governed by formal Ethics Approval and that the reporting to the survey would be anonymous. According to Edwards et al. [82], participants are more likely to respond to survey requests from universities as they know that their response would not be used for commercial benefits, rather it will be used to benefit society through research. We contacted the survey participants using the academic email of one of the authors of the paper. The time when a survey invitation is sent to the developers can directly impact their likelihood of responding to the email. Smith et al. [81] recommend avoiding sending survey invitations on Mondays, Fridays, or in December when the recipient might be out of the office. We sent the survey invitations between Tuesday and Thursday. We also posted the survey on our social media platforms (Twitter and LinkedIn). We received 175 clicks on our survey.

2.3.3 Analyzing the data

As the first step of our data analysis, we checked the completeness of our responses. Although we received 175 clicks on our survey, upon close inspection, we found 74 responses as either entirely empty responses or only including responses to the demographic questions. An additional 11 responses were 30% to 70% complete. For the sake of completeness, we left out these responses too. Hence, we

excluded a total of 85 responses from our analysis. We then assigned an ID to each response and performed our survey data analysis on the 90 fully-completed responses. However, participants only received and responded to the questions that applied to them (based on a demographic question, see section 2.3.1). For this reason, we had different sample sizes across questions. For each statistic, we mention the sample size considered for that analysis (i.e as a subscript in headings and captions).

We employed statistical and qualitative approaches to analyze the survey responses. We performed a thematic analysis for the qualitative open-ended questions. The thematic analysis involves reading through a data set and looking for patterns in meaning in order to derive themes within the context of the research question [83], [84]. In the initial iteration of the analysis, two authors of the paper independently annotated all the responses to the open-ended questions. After that, they compared the categories and discussed any ambiguities or disagreements. For the quantitative analysis, we performed statistical tests whenever applicable.

2.4 Results

In this section, we present the results by first describing the survey participants based on demographic information. Then, we present the motivation, decision criteria, and perceived challenges faced by both users and providers of GitHub Actions.

2.4.1 Survey Participants

We asked our participants three demographic questions- their experience in years, their role within their companies, and their familiarity with GitHub Actions.

Experience and Role$_{(90)}$: 33.33% of our survey participants have between one to two years of experience in the software industry. This is followed by 25.56% par-

ticipants with experience between three to six years. 17.78% of participants have less than one year of experience while 16.67% are with more than 10 years of experience. The smallest group consists of 6.67% participants who have an experience between seven to 10 years. These participants vary in their roles in a team. 64.44% (58 among 90) of the participants are *developers* followed by *managers* (14.45%) and *DevOps engineers* (8.89%). We also received responses from *Consultants* (4.44%), *Systems Engineers* (3.33%), *Testers* (2.22%), and *Security Analysts* (2.22%).

Type$_{(90)}$: We partitioned the responses following Kitchenham and Pfleegar [85] and considered three types of GitHub users.

Provider (P): Participants who have created Actions or contributed to Actions or both. We assume this group has also used Actions.

User (U): Participants who have neither created nor contributed to Actions but used Actions for one or more tasks.

Non-user (NU): Participants who have never created, contributed to or used Actions.

54.44% of our survey participants are *providers*. Among these participants, five have only contributed to one or more Action repositories (contributor) but never created an Action. 22.22% (20 out of 90) of our respondents are *users* and the remaining 21 (23.33%) participants are *non-users*. 15 out of these 21 participants said that they have not heard about GitHub Actions.

2.4.2 RQ1 - Developers' motivation for creating, publishing, and reusing Actions

We asked four questions (Q1 to Q4 in Table 2.2) to evaluate what motivates developers to create Actions, publish them on the marketplace, and use them for automation tasks on GitHub.

Creating Actions instead of reusing$_{(44)}$: While there are over 15,000 Actions in

Table 2.3: Motivations and challenges for creating Actions (P_{44})

Reason for creating Action	**% of participants**
No existing Action was available.	56.82
Existing actions are limited in functionality.	25
Existing actions are not performant enough.	11.36
Not sure how to find and reuse Actions.	4.55
Existing actions were too complex.	2.27
Check out similar Actions first?	**% of participants**
In great detail	45.45
Somewhat detail	43.18
Casually	11.36
Using self-hosted runners?	**No. of participants**
No	23
Not sure	5
Yes	**16**
Use cloud/local machines already paid for	4
Create custom hardware configurations	4
Install software from local network	3
Specific security requirements	3
OS not offered by GitHub-hosted runners	2
Challenges in creating Actions	**% of participants**
Test-ability	25
Limited resources	18.18
Privacy and security	6.82
Integration	6.82
Other	18.18
No challenges	25

existence, an increasing number of Actions are published every day on the Marketplace. This creates the question as to *why* developers prefer to develop new Actions rather than reusing the existing ones. We intend to identify the existing gaps in the current practice. In response, 56.82% of developers in our survey have stated that they create new Actions mainly because their *requirements are not met* in any of the existing Actions. One-fourth of them believe their requirement is unique and there is *no existing Action* or that *existing Actions have limited functionality* for their purposes. Developers also create Actions when the *existing Actions' performance was not good enough* (11.36%) or they are *not sure how to find and reuse Actions* (4.55%) (detailed in Table 2.3).

Table 2.4: Perceived Advantages of publishing Actions to the marketplace and why developers do not publish (P_{44})

Advantages to Publishing	**% of participants**
Visibility	27.27
Attracting contributions	27.27
Reusability	25
Accessibility	20.45
Improved code quality	11.36
Other (Get feedback, make connections)	9.09
Published Action?	**No. of participants**
Yes	19
No	**25**
Takes too much time and effort	9
Does not have permission to publish	9
Does not see benefit in publishing	5
Does not know how to publish	2

> “ The action was too complex for my task and quite unreasonable in design, so I built mine.
>
> *Survey Participant (SP)-98* ”

We know from the Marketplace trend that developers often create new Actions instead of reusing existing Actions. We are interested to understand how and to what extent the functionality and scope of similar Actions created by others impact their design decisions. Table 2.3 shows that 45.55% of developers stated that they look into others' Actions *in great detail*. 43.18% of developers said they checked similar Actions in a *somewhat detailed* fashion. Only 11.36% of developers checked similar Actions *casually*.

Publishing Actions to GitHub marketplace$_{(44)}$: There are currently 16,730 Actions on the marketplace across 21 categories. Developers in our survey stated that publishing Actions comes with many advantages such that Actions are easily *discoverable* (27.27%), *attract contributions* (27.27%), and *become reusable* (25%). Open-source collaboration often results in *improved code quality*, which is another reason (according to 11.36% participants) for publishing Actions (see Table 2.4).

> “ Publishing to the marketplace increases an organization’s visibility and attracts prospective clients/talents. It also helps an individual in building their resume. Besides, going open source can greatly improve the quality of code.
>
> *SP-19* ”

> “ Re-usability, and reduction of effort in creating new, similar workflows.
>
> *SP-80* ”

However, when asked if they published their Actions on the Marketplace, the majority (56.82%) of the responses were negative due to various reasons (see section 2.4.4).

Primary tasks that are being automated$_{(69)}$: Most developers (73.91%) are integrating GitHub Actions in their workflows for more than one task. While Actions were originally introduced by GitHub for automating CI/CD tasks within the repositories [61], [64], there is a shift in the usage direction. Only two developers explicitly mentioned *CI/CD* among their automated primary tasks. Rather, developers use Actions to automate Testing (28), Build (26), Deployment (22), and Release (19) - essentially creating their CI/CD pipeline individually and by plugging together different products. Developers also use Actions for *code review*, *linting*, and *benchmarking* (see Figure 2.2).

Key Findings: Unmet functional (for 56.82%) and non-functional (for 36.56%) requirements were the most frequently mentioned motivation of participants to create Actions. Testing has been the most frequently mentioned task that developers primarily automate using Actions. Getting more visibility and attracting contributions for better quality is developers’ main motivation to publish their Actions on the GitHub Marketplace.

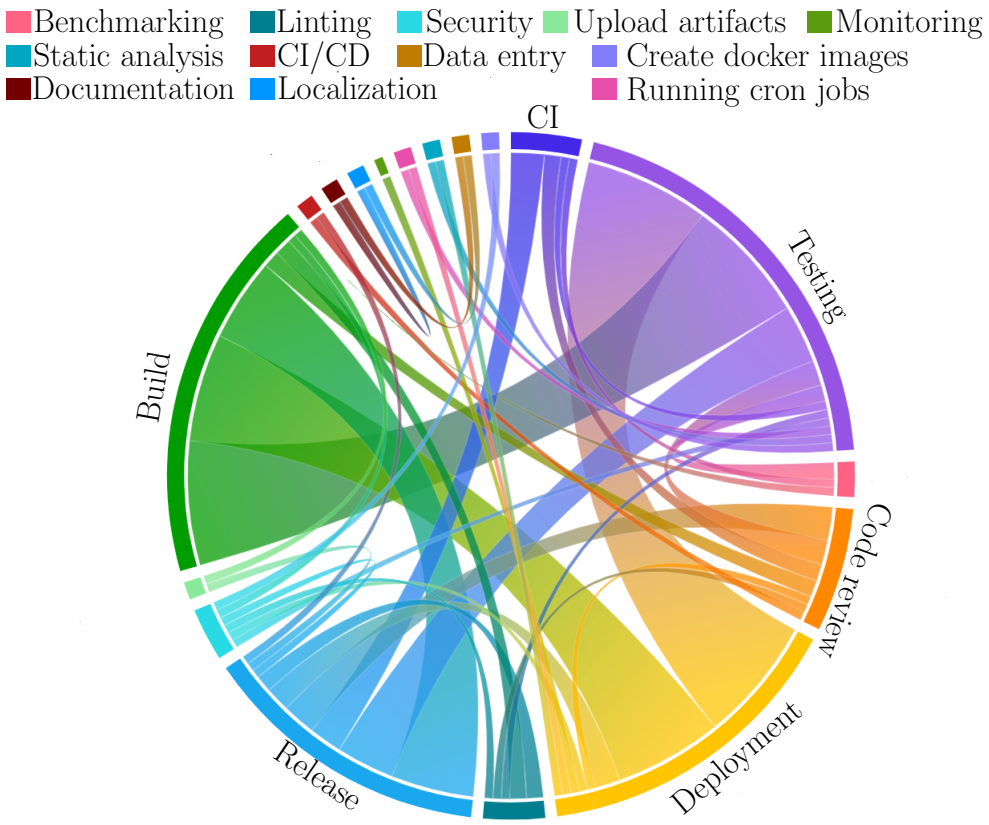


Figure 2.2: Tasks developers automate using GitHub Actions.

2.4.3 RQ2 - Decision criteria for developers to create and use Actions

Knowing why developers create and publish Actions, we now discuss *how* they do it.

When do users decide to automate tasks$_{(69)}$: The majority of the Action users (P+U) tend to automate a task in their GitHub workflow *if the task is repetitive* (95.65%) or *if the task has been automated in other repositories* (47.83%). We also observed that 33.33% of the participants automate tasks if it *requires high precision*. Our survey participants also tend to introduce automation in their workflow if a task is *mission critical*, *boring*, or *takes too long to do manually* (see Table 2.5).

> “ We try to automate anything mission-critical, where there is no room for human error.
>
> *SP-32* ”

Finding Actions for use$_{(69)}$: Knowing that 39.55% of Actions have never been published on the Marketplace [59] but are repetitively used, we are interested to identify the channels where participants identify a proper Action for their use case (Table 2.5). 68.12% of the users stated that they find Actions by *internet search, YouTube, forums such as Stack Overflow*, followed by 62.32% users who *browse GitHub Marketplace* to find Actions. Surprisingly, developers are more likely to *create their own Action* (39.13%) when they need one rather than *browsing repository-specific suggestions by GitHub* (37.68%) or even relying on *word of mouth from known connections* (20.29%). We observed that the least experienced developers (with less than a year of experience) are most likely to create their own Actions instead of searching for reusable Actions. While not a popular choice, 2.9% of users have said that they *check other repositories that have automated similar tasks*.

> “ I will identify if there are repos already created that have automated functionality that I want to automate.
>
> *S-39* ”

Table 2.5: How developers decide, find, and use Actions and the challenges they encounter (P+U$_{69}$)

Tasks developers automate	**% of participants**
Repetitive	95.65
Automated in other repositories	47.83
Requires high precision	33.33

Continued on next page

Table 2.5 – continued from previous page

Mission critical	4.35
Not requires thinking	2.9
Takes long to do manually	2.9
How developers find Actions	**% of participants**
Web, YouTube, social media, & forums	68.12
Browsing GitHub Marketplace	62.32
Creating own Action	39.13
Browsing GitHub's suggestions	37.68
Word of mouth from known connections	20.29
Checking other repositories	2.9
Reason to switch between Actions	**% of participants**
Found an Action of better quality	33.33
Created own action	26.09
Previous Action had bugs	26.09
Previous Action lacks documentation	23.19
Found an Action with more features	21.74
Found Actions with more compatibility	2.9
Never switched between Actions	40.58
How developers write YAMLs	**% of participants**
Modifying a previous workflow file	69.57
Using configurable templates by GitHub	50.72
Writing from scratch	49.28
Copying from websites or forums	43.48
Action documentation	1.45
Challenges in writing YAMLs	**% of participants**
No challenges	39.13
Never configured YAML files	10.14
Syntax	11.59

Continued on next page

Table 2.5 – continued from previous page

Debugging	8.7
Generalizability	7.25
Indentation	5.8
Missing Intellisense	4.35
Inadequate documentation	4.35
Learning curve	2.9
Other	5.8
Challenge in automating workflows	**% of participants**
No challenges	40.58
Testing & Debugging	14.49
Complexity	8.7
Inadequate documentation	7.25
API limitations	4.35
Security concerns	4.35
Lack of support	2.9
Managing artifacts	2.9
Integration	2.9
Other	8.7
Prefer not to automate certain tasks?	**No. of participants**
No	55
Yes	**14**
Preferring manual work	3
Reducing complexity	3
Cost/effort effectiveness	5
Security issue	1
Other	2

Choosing the right Action$_{(69)}$: At the time of writing this paper, there are 16,730 Actions on the GitHub Marketplace across 21 Categories while many others are

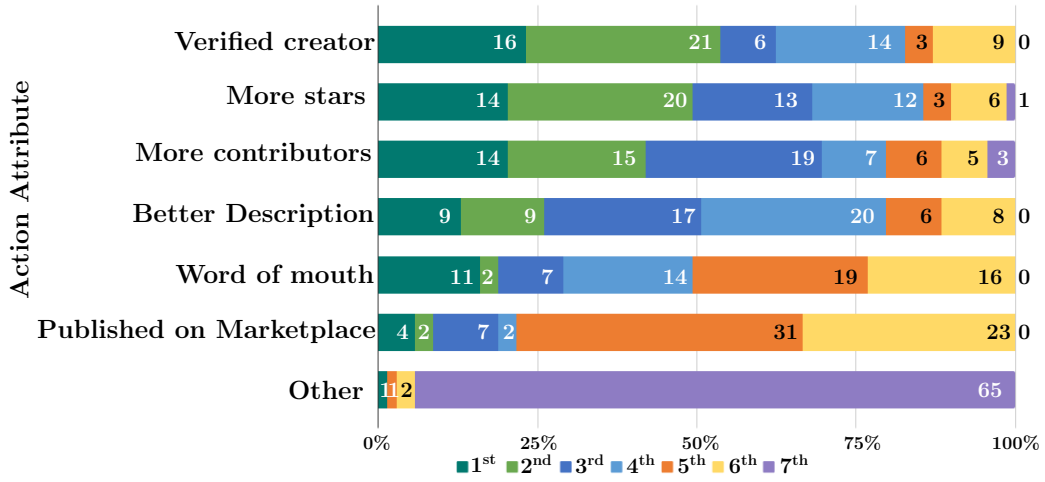


Figure 2.3: Developer preferences for choosing an Action when there are multiple Actions to choose from.

being used but not published on the platform [59]. In such a fiercely competitive ecosystem, it is a challenge to find a proper Action. We asked our survey participants how they chose an Action when multiple Actions are available for a particular task. Figure 2.3 shows the factors and their priorities to our participants. Actions having *verified creator* often have an edge over competitors. 53.62% of participants chose this factor as their first or second preference when choosing between competing Actions. This is closely followed by factors such as *more stars* and *more contributors*, each of which are chosen by 20.29% participants as their first preference. Only 21.74% participants have chosen *published on the marketplace* among their top four preferences. In fact, this factor is the most popular fifth and sixth preference among Action users. *Other* factors mentioned by the participants involved the quality of the actions and ease of code comprehension.

We studied all the results based on the varied experience groups of our participants. However, these results were rarely significantly different between the groups. 33.33% of the most experienced developers (10+ Years) chose *word of mouth* as their first preference in contrast to 15.94% developers choosing this across all

groups. On the other hand, 50% of the least experienced developers chose *better description* as their first preference compared to 13.03% developers choosing this criterion overall. Also, Action providers put a higher importance on *verified creator* while Action users prefer Actions with *more contributors*.

Switching to a different Action$_{(69)}$: In this competitive ecosystem, we were interested to learn the frequency and users' decision criteria for switching to alternative Actions. 40.58% (28 of 69) of our respondents stated they never switched between Actions. This can be due to the lack of experience as 26 of these respondents have less than seven years of experience. The others have switched to a different Action because they found an Action with *better quality* (33.33%), *more features* (21.74%), or *better compatibility i.e. regular updates* (2.9%). Users also switch to a different Action when their previous Action *have bugs* (26.09%) or *not properly documented* (23.19%). Interestingly, churningout due to in-house development of Actions was the second-most popular reason for churning out (Table 2.5). Developers with 10+ years of experience most frequently switch Actions for this reason.

> “ Another action kept providing updates while the original action lost its compatibility.
>
> *SP-69* ”

> “ Action stopped supporting my required use case.
>
> *SP-32* ”

> “ We decided to automate our code review and my senior developer had a suggestion for a code review action which was reviewing and merging pull requests in one.
>
> *SP-98* ”

Writing workflow (YAML) files$_{(69)}$: Users add a YAML script to their repository for adopting and using an Action and configuring automation for their workflows.

The success in adoption and proper use of any Action depends on these files. Among our participants, *modifying a previously used workflow file* (69.57%), and *using configurable templates by GitHub* (50.72%) are the most popular methods for writing such files. Other than that, oftentimes users *write from scratch* (49.28%) or *copy from websites or forums such as Stack Overflow* (43.48%) (see Table 2.5).

GitHub-hosted vs self-hosted runners$_{(44)}$: GitHub Actions are dependent on runners to execute Action workflows. One can host their own runner and customize the environment for Action workflows (self-hosted runners) or use GitHub virtual machines (GitHub-hosted runners). 36.36% (16 out of 44) Providers (P) in the survey stated that they use self-hosted runners (see Table 2.3). Developers use self-hosted runners because they want to *use machines they already paid for*, *create custom hardware configurations*, *install software from local network*, *use an operating system not offered by GitHub-hosted runners*, or because they *have specific security requirements*.

Why developers do not automate certain tasks$_{(69)}$: To have a better picture of the developers' decision for automating development tasks, we asked users whether they prefer *not* to use Actions at times. Only 14 out of 69 (20.28%) respondents preferred not to automate certain tasks. Half of these developers (7 out of 14) had more than 10 years of experience. Developers prefer not to automate tasks when the task is not worth the cost/effort that goes into using Actions (5), or when using Actions increases the complexity of their projects (3), poses security risks (1), poor documentation (2), or simply because they prefer to perform certain tasks manually (3) (Table 2.5).

> " I don't use Actions when repositories are sensitive and I don't want third parties having access.
>
> *SP-76* "

Key Findings: The abandoned number of alternative Actions provides software developers with multiple options for automating their tasks and the majority (79.71%) wish to automate all the development tasks. Our study shows that most often the developers tend to automate the repetitive tasks within their software repositories and then either use online (web) search or the GitHub marketplace to find an Action. In this context, the unfulfilled non-functional requirements (65.32%) and the development of in-house solutions (26.09%) motivate developers to migrate from one Action to another.

2.4.4 RQ3- Challenges for developers in creating, publishing, and using Actions

Further, we investigated developers' challenges in creating, publishing, and configuring Actions.

Creating Actions$_{(44)}$: 75% of participating developers faced one or more challenges in developing Actions. The most frequently mentioned challenge was the *testing ability* of Actions where 25% of the developers faced difficulty to test and debug their Actions (Table 2.3). According to 18.18% participants, *limited resources* i.e. lack of documentation and support makes onboarding difficult for new developers. Developers also have *privacy and security concerns* (6.82%) as well as issues with *integrating Actions into workflows* (6.82%). Developers also pointed to the lack of proper monetization channels, the poor design of the Actions framework, limited GitHub APIs, and slow updating of GitHub runners (where Actions run) as challenges in creating Actions.

> " GitHub Actions fail too frequently (especially with Windows runners) with no clear explanation, and these errors are frequently not reproducible. Additionally, GitHub is too slow with adding new macOS versions to runners.
>
> *SP-71* "

Publishing to the Marketplace$_{(44)}$: Not all Actions are available on the marketplace. The majority, 56.82% (25 out of 44), of the Action providers (P) in our survey

Table 2.6: Debugging workflow (YAML) files and using GitHub Marketplace (U_{25})

Debug YAMLs by	**% of participants**
Check Q/A forums such as Stack Overflow	92
Create a pull request/bug report for Action	36
Contact the Action developers	28
Ask on social media such as Reddit/Twitter	24
Problem with marketplace	**% of participants**
No challenges	60
Searching for products	16
Hard to quality check	8
Less marketed	4
Does not use Marketplace	12

did not publish their product on the Marketplace. Nine out of these 25 developers reasoned that *publishing needs much time/effort* while other nine developers *did not have permission to publish*. Quite interestingly, the remaining five (out of 25) developers *did not see the benefit in publishing* while two said they *did not know how to publish* (Table 2.4).

Configuring workflow (YAML) files$_{(69)}$: Configuring a workflow using a YAML file is not a trivial task. 50.73% of Action users reported challenges in configuring their workflow files. 11.59% of the developers have said that YAML file's *syntax is confusing and error-prone* and 8.7% developers have commented that YAML files are *difficult to test and debug*. Another 5.8% of the participants *struggle with indentation rules*. This is while others were challenged with the *missing intellisense* (i.e. code completion, syntax highlighting) and *inadequate documentation* about how to configure Actions using YAML (Table 2.5).

> " I am very used to having intellisense when I write code with any language, so it was a bit challenging to write YAML without intellisense help on GitHub.
>
> *SP-11* "

> " YAML is a bad for writing configuration code. It's untyped, which frequently leads to serious bugs in configuration code. I wish there was a statically-

typed and more reliable alternative to YAML available and officially supported by GitHub Actions.

SP-71 ”

“ YAML is great to read, and a nightmare to write.

SP-72 ”

We asked survey participants where they look for information when facing an issue with YAML configurations. 92% of the participants use *Q/A forums such as Stack Overflow* while 36% users *create a pull request/bug report in the Action repository* (Table 2.6).

Automating workflows using Actions$_{(69)}$: 59.42% of users experienced one or more challenges in automating workflows using Actions. Users had issues as the workflows are *hard to test and debug* (14.49%), *too complex for some tasks* (8.7%), and *have inadequate documentation* (7.25%) (Table 2.5).

“ A typical day of working on GitHub actions is a pain. Having to make your changes, hope it works, commit, push, launch the action, wait 10 min, see it fail, continue.

SP-73 ”

“ There is a lack of tutorials and guides for GitHub Actions. It’s frequently not clear how to achieve a certain configuration, and then it’s quite hard and time-consuming to debug.

SP-71 ”

Using the marketplace$_{(25)}$: We asked the users and contributors if they faced any challenges in using the GitHub Marketplace. Seven out of 25 participants stated it is *hard to search for products* and *hard to quality check*. It implies that a review-based

system such as the mobile app stores for GitHub Marketplace would be helpful to the marketplace users. The marketplace currently shows the number of installs for Apps and the number of stars for Actions, but these are not helpful metrics for checking the quality of products. Other than that, one user was not aware of GitHub Marketplace for a long time and felt that the marketplace was *not marketed enough* (Table 2.6).

> “ Marketplace does not provide an effective way to filter and sort based on quality, version, contributions, etc.
>
> *SP-84* ”

Key Findings: Test-ability is the most frequently reported problem for creating and using Actions. The YAML programming language which is used to develop Action's configuration files was reported as buggy and hard to test. Unfamiliarity with the syntax, not having code IntelliSense, and inadequate documentation about configuring Actions using YAML are the top challenges GitHub users face with configuring workflows. Publishing on the Marketplace is also time-consuming and developers refrain from it.

2.5 Discussion and Recommendations

Implication For Researchers: It only took GitHub Action 18 months to surpass Travis CI, which had dominated the CI scene for over nine years [61]. Many software tools are either being replaced by Actions or being integrated into and packaged as Actions. Researchers can examine how the invention of Actions has changed the ecosystem for software tools. So far, the research community has only looked into the Actions from the users' perspective, i.e. they systematically crawl repositories that are *using* Actions (see Table 2.1). The field of automated software engineering has been growing extensively. Distinguishing between the providers and adopters is essential since Actions in their nature are atomic building blocks of automation in practice. The actions provide a unique opportunity for researchers

to evaluate the tasks that are being automated and the users' feedback, usage, and preferences within practice [86], [87]. In this study, we have highlighted the challenges developers face in automating workflows using Actions, such as Actions being hard to test and debug, missing IntelliSense for configuration files, or having integration and security issues. This also provides new opportunities to explore the field of programming languages, and mining software repositories.

For the GitHub Actions team: 21 of our 90 participants do not use Actions, only six of them know that Actions exist, but half of them are unaware of the benefits of Actions. One respondent said:

> " I am not sure of the possibilities.
>
> *SP-59* "

Limited resources is the second-most perceived challenge by Action creators (Table 2.3). Action users mention *inadequate documentation* and *lack of support* as two of the challenges in automating workflows. Although GitHub has extensive documentation for GitHub Actions[3], it seems many practitioners are either unaware of them or find them unhelpful. We asked developers about the challenges they face in using the Marketplace. One of the responses was:

> " The marketplace isn't as relevant, it took me so long to discover it even exists, my developing career would be so easier if I did earlier.
>
> *SP-11* "

62.32% of the participating developers search Marketplace to identify a proper Action for their tasks. This is while only 21.74% of them state any preference in using the Actions published on the Marketplace. This implied the power of Marketplace as a hub of gathering information but the characteristics of a multi-sided

[3]https://docs.github.com/en/actions

platform are missed from the Marketplace which led to multiple trust and reuse challenges [29]. Implementing stronger mechanisms as such to assist developers in their decision-making is recommended.

Nine out of 25 (36%) developers who did not publish their Action on the marketplace said publishing *takes too much time and effort*. We also recommend GitHub make publishing Actions to the marketplace less strenuous while also ensuring that the process stays effective enough to enforce the best practices. GitHub Marketplace should introduce reviews similar to mobile stores that users can benefit from community feedback. To the least, Marketplace should let users filter Actions based on metrics such as the number of contributors, verified creator, etc.

For Action providers: Action providers' #1 reason for creating new Actions instead of reusing existing ones is the unavailability of the needed functionality for a particular task. We found out that 39.13% developers create their own Action when they require one while only 2.9% developers check if similar Action was available in other public repositories (Table 2.5). Also, the second-most common reason why developers switched to a different Action is that they created their own Action. To leverage the re-usability to the fullest extent, we recommend that developers perform a thorough analysis of existing Actions before creating their own. One alternative to creating would be to contribute to an existing Action, which is likely to save time and improve re-usability. We also recommend that developers publish their Actions to the Marketplace so that they get better visibility, more contributors, and improved code quality. Furthermore, the structured status of YAML files and the openness of the Action repositories makes it quite possible to use data analysis tool to mine and compare the functionalities within the existing Actions.

For Action users: There are currently over 15,000 Actions across 21 categories in the GitHub Marketplace. It is difficult for potential Action users to decide which Action to use. *Security concerns* is one of the five major developer challenges when

automating workflows on GitHub (see Table 2.5). In order to avoid accidentally using a compromised Action that might have had its code altered and might potentially be used to steal secrets, GitHub suggests referring to reusable Actions using their specific commit SHA[4]. As Decan et al. [60] pointed out, this recommendation is not followed in practice. Our survey showed that the developers who are using an Action prefer *verified creator*, *more stars*, and *more contributors* when they are choosing between Actions. To avoid security issues, GitHub recommends only using actions whose creator can be trusted. We recommend that developers follow GitHub's security guides[5] for GitHub Actions.

Although Borges et al. [88] warned about relying on the number of stars as a metric for repository quality, stating that more stars could mean effective marketing and advertising strategies and not necessarily solid software engineering principles and practices. However, the number of stars remains a popular metric among practitioners for evaluating repositories. Finally, more contributors indicate a project is more active and better maintained. Therefore, we back our survey findings and recommend potential Action users consider verified creator, more stars, and more contributors when they are choosing between Actions.

2.6 Threats to Validity

External Validity: We used *closed and open invitation* strategies for our survey [89], [90]. Closed invitations allowed us to have more control (such as inviting Action creators). We further sent open invitations via social media channels. To have a better understanding of the participants, we grouped them based on their familiarity with Actions as well as their years of experience. We present our analysis based on 90 complete responses, which should be generalized only in consider-

[4]https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions
[5]https://docs.github.com/en/actions/security-guides

ation of these demographics and sample size. Nothing in the study specifically limits replication on a bigger group of developers to make our conclusions more robust.

Construct Validity: We used thematic analysis for the open-ended questions. The accuracy of this analysis depends on the annotators' accuracy and perception of classification. The exploratory design of the study could have introduced *researcher bias*. To reduce this, annotators worked independently and within iterations. Further, any disagreement was discussed and mediated.

Internal Validity: We designed our questionnaire in iterations where the initial draft was tested with a small group for *alternative form reliability* and *content validity*. Also, a later version was reviewed and tested for clarity, bias, and ethics by an independent party. Yet, there is a chance that some survey participants interpret the questions in different ways and find them unclear. Another threat could arise from the arrangement of the options in a ranking-based question which biases participants in choosing top-listed options. We ordered the questions in a natural sequence and in an effort to clarify the context.

Conclusion Validity: We draw conclusions in our study based on a survey, which can be inaccurate at times. Surveying software developers do not always provide a comprehensive perspective of real-world practices [91]. However, surveys have been prevalent as a research tool in software engineering. We see the survey as a complementary tool for other types of study such as mining.

2.7 Conclusion

Despite the inherent interest of the research community in software automation, there is not much known about the motivation and decision criteria of the Action adopters. So far, the research community has focused on the quantitative anal-

ysis of GitHub repositories to investigate the usage and adoption of GitHub Actions. This study aims to complement the current literature and look beyond the statistics mined from repositories. Through a qualitative survey with developers, we present the motivation, decision criteria, and challenges in creating, publishing, and using GitHub Actions. Our results demonstrate the limited leverage of GitHub Marketplace as a multi-sided market. Compared to what is seen in the mobile app stores, developers refer to the marketplace as an aggregator and for searching Actions. Rather, marketplaces should connect users and facilitate feedback loops. Our findings help Action providers identify and improve areas to improve their competitiveness. We believe this study also helps researchers understand the status quo and conduct relevant research to facilitate automation practices on software repositories such as GitHub. We found that configuring and debugging the workflow YAML files is one of the most prevalent problems with using Actions. An automatic workflow generator framework can help alleviate this problem and make GitHub Actions more accessible to a larger audience.

Chapter 3

GitHub Marketplace for Practitioners and Researchers to Date

Abstract

GitHub, known primarily for its version control functionality through Git, launched its own marketplace in 2017. GitHub Marketplace hosts third-party apps and actions to automate workflows in software teams. The GitHub Marketplace provides a unique source of information about the tools used by practitioners for automation in the Open Source Software (OSS) ecosystem. Automating development processes in the OSS ecosystems has also been the subject of many research studies. This paper presents (i) the very first comprehensive mining of the GitHub Marketplace, (ii) the systematic mapping of literature in automation in OSS and, (iii) a multifaceted comparison between tools used in practice and the tools developed by researchers. We performed quantitative and qualitative analysis of 440 Apps and 7,878 Actions across 32 different categories in the marketplace on their provider, platform, and user data. We also systematically gathered 515 research papers between 2000 to 2021 in OSS automation and classified them according to the automation categories, type of research, type of analytical tools, and their reproducibility. The results of our comparison demonstrate the popularity of automation tools for Continuous Integration and Utilities among practitioners. Code

Quality and Testing are the most popular automation domain among academics. We also found that while researchers are highly focused on experimentation (51.5% of the papers), the practitioners are mostly using exploration tools (67.2% of the products) to identify ad hoc anomalies. This is whilst only 20.56% of the OSS automation tools published in academia are available, which compares with consistent and convenient access to the practitioners' tools through the GitHub Marketplace. Our study's results help researchers close the gap between industry and practice and show the path for further mobilizing our knowledge. Our results help researchers build on top of the existing work, identify the practical relevance of their research, and help practitioners navigate technology and find synergies.

3.1 Introduction

The software engineering community has been offering automation tools for development teams. With the urge to help practitioners with their tasks, researchers publish, develop tools, organize conferences, and discuss automation in software engineering. Recently, GitHub offered a marketplace to gather and share automation tools for GitHub-hosted projects. Third-party developers provide tools on GitHub Marketplace to automate workflows in software repositories. The majority of these tools are open source. This provides us with the unique opportunity to compare the alignment of our research effort with what is being used and offered in practice to improve our knowledge mobilization efforts [92]. The software engineering research community has also been concerned with the automation of development workflows and providing decision support to assist software teams in improving team productivity and product value. Yet, the GitHub Marketplace has never been the subject of an investigation to evaluate the alignment or difference between research efforts (the published studies) and the practice (GitHub automation tools).

Software marketplaces have been studied since 2012 from the engineering perspective. Harman et al. [32] provided an insight into the importance of the marketplace ecosystem for software engineering practices by investigating the mobile app stores. Martin et al. [33] synthesized the research and development directions in mobile app stores. Mobile app stores enable developers to distribute content while establishing a communication channel between users and developers via ratings and reviews [33]. The results of Martin et al. synthesis in 2017 showed that the research in mobile app stores covers a spectrum of analytics ranging from code to users. Similar to mobile app stores, GitHub automation tools (apps and actions) on the marketplace also have technical and non-technical attributes.

GitHub Marketplace is a software repository that allows developers to provide free or paid tools to GitHub users for automating their workflows[1], in particular, when hosted on a GitHub repository. Apps and actions are the two types of tools in this marketplace. While both provide ways to build automation and workflow tools, they each have differences[2]. **GitHub actions** are designed for automation of customized individual tasks within a workflow[3] including the automation for CI pipelines, building, testing or deploying software applications. They are written in `YAML` files and can be built as a `Docker` container or a `Javascript` snippet. GitHub actions are free for users. **GitHub apps** are applications that GitHub users can install on user or organization accounts. Apps can access certain repositories when explicitly specified and allowed by the user[4]. Apps can be free or paid, or can have multiple paywalls depending on their functionality. Apps and Actions are categorized by their developers into up to three categories. These categories are predefined by the marketplace and describe a high-level objective of the tools.

[1]https://docs.GitHub.com/en/developers/GitHub-marketplace/GitHub-marketplace-overview/about-GitHub-marketplace

[2]https://docs.GitHub.com/en/actions/creating-actions/about-custom-actions#strengths-of-GitHub-actions-and-GitHub-apps

[3]https://docs.GitHub.com/en/actions/creating-actions/about-actions

[4]https://docs.GitHub.com/en/developers/apps/getting-started-with-apps/about-apps

In this context, we answer three research questions (RQs):

RQ1- What category of tools are offered for automating development workflows, and how do they compare in nature and popularity?

Why and How: To provide a better understanding of the attributes and data structure within this repository, we mined the characteristics of apps and actions in the GitHub Marketplace and explained the status quo using all the available fields in the repository and performed a comparative study wherever appropriate.

RQ2- How have the automation tools for open-source software evolved in the research community?

Why and How: We systematically gathered literature on automation tools and techniques within the open-source ecosystem and compared the papers over the years. Further, we mapped the topic of these academic studies into the marketplace tool categories to enable comparison between research and current practice.

RQ3- How do the automation tools in the marketplace compare with software engineering literature in nature and popularity?

Why and How: We compared the state of the art with the state of the practice. To this end, we first gathered app and action information and extracted a list of categories and frequency of coverage from the GitHub Marketplace. We then performed a systematic literature analysis and mapped the papers retrieved from our systematic search into the marketplace categories and discussed the gap and trends.

The gap between academic research and industry practice in software engineering has long been a matter of concern. Scholars have made various efforts to tackle

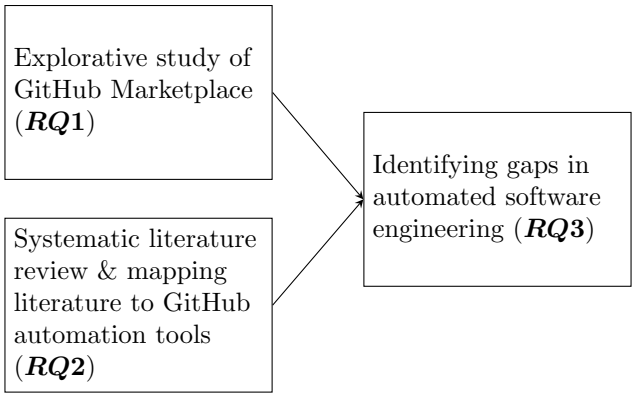


Figure 3.1: Relation between the Research Questions in this study

this issue, including pedagogical endeavors [93], fostering collaboration through mutual data exchange [94], [95], and improving feedback and collaboration processes [96]–[98]. Different authors took a variety of approaches to analyze and address this gap. For example, Santos et al. [99] explored the differences between academic researchers and practitioners in software testing, Schmid et al. [100] conducted interviews with practitioners and researchers from the SPLC[5] community to reflect on this, Ferreira et al. [101] showed this gap between software maintenance techniques proposed by researchers and those used by practitioners. Further, Garousi et al. [102] highlighted the need for software engineering research to be relevant to industrial practice and discussed the challenges faced by the community in achieving this. While introducing the GitHub Marketplace, we take a novel perspective to quantify the gap between research and practice in the context of automation in OSS.

The widespread popularity of GitHub for hosting software repositories and managing product releases, along with the abundance of workflow automation tools, highlights the relevance of GitHub Apps and Actions in addressing real-world software engineering challenges. These tools serve as a unique data source, enabling us to observe practices, uncover pain points, and identify trends in soft-

[5]https://dl.acm.org/conference/splc

ware automation. Our study involves comparing research studies in the software engineering community with the automation tools published on GitHub Marketplace. This comparison helps us explore the gap between research and practice in open-source software tools. While it is important to acknowledge that not all GitHub Actions are available on the marketplace, this platform still offers a representative collection of these tools. It is worth noting that our objective is not to compile an exhaustive list of all papers or open-source tools discussing automation in the software product life cycle. Rather, our aim is to provide a fair depiction of the state of the art through systematic literature reviews and the state of practice by mining the GitHub Marketplace.

Section 3.2 presents the related work. Section 3.3 describes the research methodology, including data extraction, synthesis, and analysis. Section 3.4 presents the observations from mining the GitHub Marketplace. Section 3.5 shows the results of the systematic mapping study. Section 3.6 provides the gap analysis. Section 3.7 provides discussion and future implications. Section 3.8 discusses the threats to validity. Section 3.9 concludes this paper.

3.2 Related Work

GitHub Marketplace as a distribution platform of automation tools nor as a software repository has not been discussed in the literature.

GitHub Marketplace includes apps and actions. We identified seven studies which discuss workflow automation by mining the repositories which are using GitHub actions. **None** of these studies have pointed to the GitHub Marketplace or used it for data collection. In what follows, we briefly refer to the work done in software engineering on the analysis of the marketplace and then discuss the mining studies focused on GitHub actions.

3.2.1 Mining GitHub Workflows

We could identify seven major studies analyzing **the use** of GitHub actions for workflow automation. These studies retrieve a sample set of repositories and mine the `YAML` files to understand how GitHub actions have been adopted by the users. To this end, they randomly sampled a set of repositories and filter out the repositories that do not have action configuration files (`YAML` files). None of these studies looked into the GitHub Marketplace as a software repository or as a means of sampling.

In Study 1, Kinsman et al. [103] reported that developers commonly post issues on GitHub asking for assistance with GitHub actions. They further used the `RepoReapers` dataset [104]. As the result of mining 3,190 of such GitHub repositories, they showed that the adoption of actions increases the number of rejected pull requests and decreases the number of commits in merged pull requests. In Study 2, Wessel et al. [63] examined 1,489 Projects, and they found that 29.8% of their sample adopted GitHub Actions, mainly to automate tasks related to utilities, continuous integration, and deployment. The authors reported that the adoption of GitHub Actions led to a longer wait time before a pull request was accepted and an increase in the number of rejected pull requests. They also found that these rejected pull requests had more commits while having fewer comments. Furthermore, the authors looked into how the communities discuss GitHub Actions and found that developers frequently use GitHub Issues to ask for help with GitHub Actions. Similarly in Study 3, Chen et al. [64] looked into how GitHub actions affected the frequency of commits, pull requests, and issue resolution effectiveness. They explored 6,246 Projects on GitHub and discovered that adopting actions decreases commit frequency, increases closed issues, and decreases the number of pull requests. However, unlike [63], the authors found that adopting actions reduced the pull-request resolution latency. In Study 4, Decan et al. [105] studied

67,870 Repositories for the reuse of actions. They found that the majority of them were distributed by GitHub and belonged to the 'Utilities' or 'Continuous integration' categories.

While using different tools to clone repositories, all these four studies used a similar approach for retrieving the usage of Actions. They considered a repository being automated by Actions if they could find a `YAML` file in the `./github/workflows` directory of a GitHub project. By this, they focus on the usage of automation tools and the tool users' adoption process.

In Study 5, Golzadeh et al. [61] focused on continuous integration (CI) and analyzed how the CI landscape had changed since GitHub actions were introduced. As the authors studied how frequently CIs were being replaced by alternatives, they found that GitHub actions attracted most of the migrations and took only 18 Months to overtake Travis CI in popularity. In this study, authors manually analyzed documentation to find the path for `YAML` files. In Study 6, Mazrae et al. [106] studied the migration of developers from other CI/CD tools to GitHub actions. They interviewed 23 Software practitioners and reported that developers were gradually moving to use GitHub actions for better integration to GitHub, better support of different platforms, and more reliability. In Study 7, Valenzuela-Toledo et al.[65] examined types of GitHub action workflow modifications made by developers in 10 Popular GitHub repositories. They manually reviewed 222 Commits pertaining to workflow alterations and discovered 11 distinct categories of changes.

In Study 8, Benedetti et al. [75] investigated the security issues affecting action workflows and implemented a methodology (`GHAST`) for automatically assessing the presence of security issues in workflows. The authors used `GHAST` against 50 Open-source projects to provide an overview of the current security landscape of action workflows. Lastly in Study 9, Koishybayev et al. [76] created a workflow

auditing GitHub actions, `GWChecker`, to help mitigate common security errors in the `YAML` configuration for CI/CD workflows. The workflow files are audited by `GWChecker` for plaintext secrets using regular expressions, versioning tags, non-verified actions or actions that are not available on the marketplace, and insecure triggers.

3.2.2 Mining Marketplaces in Software Engineering

To the best of our knowledge, there is no existing literature on mining the GitHub Marketplace within the software engineering field.

In recent years, app marketplaces have drawn a lot of attention among researchers within the software engineering community. Harman et al.[32] pointed out that app marketplaces were different from traditional ones (for example, they do not have source code), and provided a framework for app marketplace analysis. Their framework consisted of extracting and refining data, extracting feature information from textual descriptions, as well as computing metrics for analysis such as correlation analysis. They also analyzed the technical, customer, and business aspects of some apps in the BlackBerry app store. Chen et al. [107] proposed `ARMiner`, a novel framework for app review mining, in order to extract valuable information from user reviews with minimal human effort. After filtering out non-informative reviews using a pre-trained classifier, they employed topic modeling and a ranking scheme to automatically prioritize the informative reviews. Krüger et al. [108] described a preliminary analysis of the Eclipse Marketplace with a view of providing a glance at open marketplaces as well as initiating further research. They proposed to mine marketplace data to address questions such as who contributes to successful plug-ins, in order to identify leading developers and communities, leading to collaborations and new research directions. Li et al. [109] performed a performance and usability evaluation of five open-source IDE plugins that identify

and report security vulnerabilities. They investigated the types of vulnerabilities these plugins could detect, the quality of detection, and the user-friendliness of plugin outputs. They found a mismatch between the claimed and actual coverage of the plugins, while most plugins had a high false-positive rate, and were not developer-friendly.

GitHub Marketplace is a platform where developers provide automation tools (apps and actions). This marketplace has never been researched in software engineering. This is despite the seven mining studies that were conducted on a random or a convenient sample of repositories which are using actions for automating their processes.

3.3 Research Methodology

Our goal is to identify the alignment and gaps between automation tools for open-source repositories in practice and the ones offered by the research community. Hence, our research methodology consists of three major parts. First, for **RQ1**, we used empirical protocols to retrieve data and mine the GitHub Marketplace software repository. We mainly used statistical methods to introduce this software repository and derive observations. In **RQ2**, we performed a comprehensive literature analysis to extract the spectrum of software engineering research for automation tools in open source. To make the results of our literature study comparable with **RQ1** and perform a gap analysis, we used the GitHub Marketplace categories as the schema to build a structure around the literature. Finally, for **RQ3**, we compared the state of the art with the state of the practice by using our findings from the previous RQs.

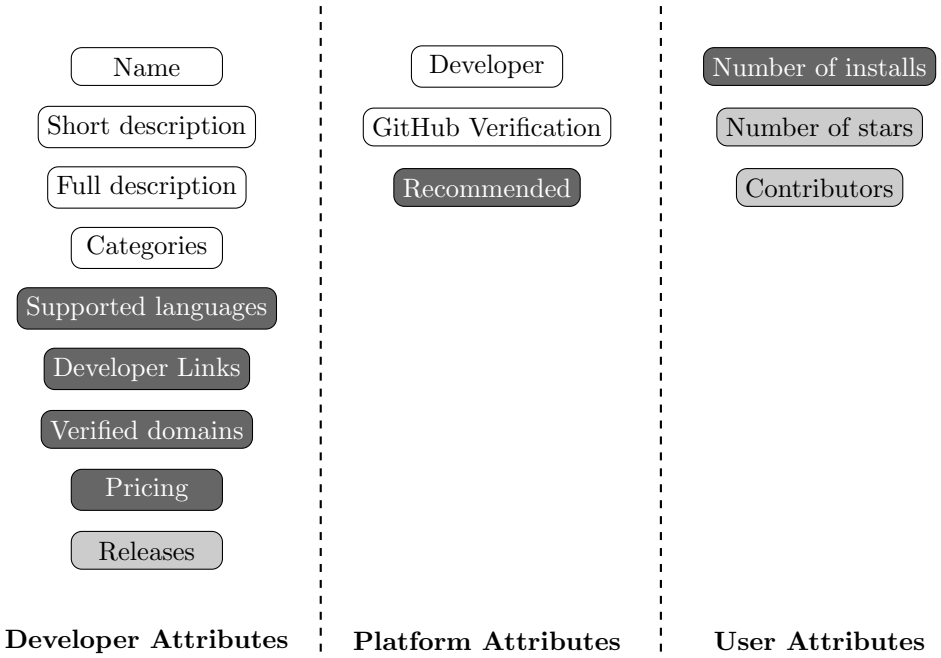


Figure 3.2: Attributes on the GitHub Marketplace for apps (dark gray), actions (light gray), or both (white). Each attribute is either defined by the developer, platform, or user.

3.3.1 Process of Gathering Data from the Marketplace

To gather apps and actions from the GitHub Marketplace, we used `Selenium`[6] (for automated loading of the JavaScript components) along with `Scrapy`[7] (as the main web page crawler). Each web page loads only 1,000 Apps or actions. While this had no implication for app data (since there are less than 1,000 Apps in the marketplace), this restriction meant that we could extract only a fraction of the action data from a static page.

To avert this issue, we designed a set of knitted crawlers that not only started from the marketplace's main page but also took each action category as a scrapping root. For most categories, there were less than 1,000 Actions on the marketplace, thus we did not face any issues in gathering all the information. However, for four

[6]https://www.selenium.dev/
[7]https://scrapy.org/

categories, there were more than 1,000 Actions. As a result, we used the embedded marketplace data sorting options- 'Best Match', 'Recently added', and 'Most installed/starred'. Each sorting method displayed a different set of data, allowing us to store additional actions not previously found.

Apps' and Actions' detail pages differ from each other. Also, actions have a free outline and style using GitHub tools which each developer can design. To ensure that we are gathering all the existing metadata for apps and actions, we parsed through the different `HTML` header layers for apps and actions. As developers provide different levels of detail about each action, we created a unified list of all the data entries provided across different actions on GitHub.

We could uniformly gather 11 Action attributes and 13 App attributes across the marketplace. Figure 3.2 includes details of these attributes. Some of the apps and actions had invalid URLs to their detail page, resulting in missing details for those items. However, we kept a record of such items in our data collection process. Figure 3.4 shows the categories for apps and actions. The results of this analysis are presented in Section 3.4.

3.3.2 Protocols Used for Systematic Mapping Study

To answer **RQ2**, we conducted a systematic mapping study with a view of comparing current trends in the open-source research community with the ones in practice. To perform this systematic study, we followed the guidelines of Peterson et al. [110]. In its nature, systematic mapping is aimed at performing general analysis of the topics and trends in the field, which perfectly serves the purpose of comparison between the state of the art and the state of practice we formulate in the **RQ3**. Figure 3.3 shows an overview of this systemic mapping. In what follows we explain the protocols of the mapping process which resulted in manually analyzing 515 papers in this study.

Source Selection

Our objective was to map the state-of-the-art software engineering to the current categories of apps and actions in the GitHub Marketplace. As such, we chose broad search terms to gather all the papers focusing on open-source software engineering. Our search query was as follows:

("Software Engineering" OR "SE") AND ("GitHub" OR "OSS" OR "open source" OR "open-source.")

Following previous studies, we gathered papers from five publication sources, namely ACM Digital Library, IEEE Xplore, ScienceDirect, Scopus, and Inspec. To ensure search quality, we applied two broad *Inclusion criteria* while performing a search in these libraries:

Inclusion Criteria I: Studies must have been published between 2000 and 2021,

Inclusion Criteria II: Studies must have been published in journals or conference proceedings. Overall, our initial search resulted in 11,741 Publications.

Overall, our initial search resulted in 11,741 Publications. We used a set of heuristic exclusion criteria and narrowed down the search. We excluded the below papers:

Exclusion Criteria I: Studies not presented in English,

Exclusion Criteria II: Duplicate studies,

Exclusion Criteria III: Studies that do not have *software* and its variations in title, abstract, keywords and *automation* and its variations in title, abstract, and keywords,

Exclusion Criteria IV: Studies having less than six pages,

Exclusion Criteria V: Studies not presented in the top 20 Publication venues according to Google Scholar's ranking[8],

Exclusion Criteria VI: Studies providing a systematic review of the literature.

[8]https://scholar.google.ca/citations?view_op=top_venues&hl=en&vq=eng_softwaresystems

Table 3.1: Selected publication venues from Google Scholar as used for exclusion Criteria V and the final number of papers studied from each venue.

Rank	Publication	#Papers
1	ACM/IEEE Inter. Conference on Software Engineering	63
2	Journal of Systems and Software	99
3	IEEE Transactions on Software Engineering	101
4	Information and Software Technology	99
5	ACM SIGSOFT Inter. Symposium on Foundations of Software Engineering	28
6	Empirical Software Engineering	32
7	IEEE Software	12
8	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	1
9	Mining Software Repositories (MSR)	9
10	IEEE/ACM Inter. Conference on Automated Software Engineering (ASE)	49
11	ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	0
12	Inter. Conference on Software Analysis, Evolution, and Reengineering (SANER)	4
13	Proceedings of the ACM on Programming Languages	1
14	Software & Systems Modeling	3
15	Inter. Symposium on Software Testing and Analysis	3
16	IEEE Inter. Conference on Software Maintenance and Evolution	5
17	Software: Practice and Experience	4
18	Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	0
19	IEEE Inter. Requirements Engineering Conference	2
20	ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)	0
	Total	515

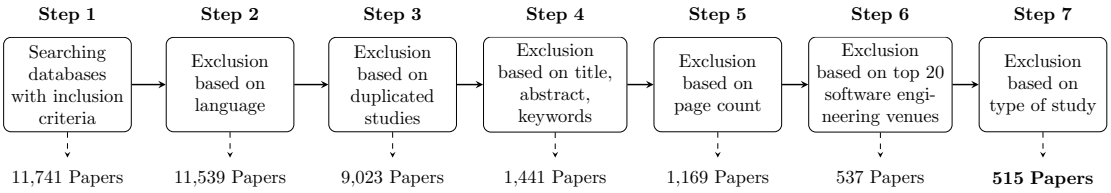


Figure 3.3: The process of our systematic mapping with the number of papers retrieved at each step.

We retrieved a total of 515 Papers after applying the exclusion criteria. The details of the number of publications are provided in Table 3.1.

Data Extraction and Synthesis

We performed three types of classification for the papers in our systematic litreture review.

First, we used the schema of automated tools and analytics provided by Buse and Zimmermann [111] to categorize each paper. They introduced nine different types of analytical tools for development activities. They categorize the analytical tools as exploration, analysis, or experimentation, while each tool can fall under past, present, or future. We identified the analysis type of a paper by the time frame it pertains to (Past, Present, Future) and the employed technique (Exploration, Analysis, and Experimentation), as laid out by Buse et al.[112]. Trends, Alerts, and Forecasting find important conditions by exploring past, present, and future, while Summarization, Overlays, and Goals explain conditions in the same time dimension sequence. Finally, Modeling, Benchmarking, and Simulation compare alternative conditions in the past, present, and future.

Second, we used the taxonomy proposed by Wieringa et al. [113], which contains five categories evaluation research (problem investigation in practice), solution proposal, validation research, philosophical paper (offering a framework), opinion paper, and experience paper (discussion of an author's personal experience). We classified a paper into more than one research type wherever applicable.

Further, we created a schema for the availability of the artifacts in these studies. We evaluated if the automation tool or the data used for experimentation was available or not. We also checked if the links were deprecated or not. As a result, we tracked each paper that contained data and/or code and gathered the links if available.

Third, for a side-by-side comparison of the topics of automation in the open source ecosystem in practice and in literature, we classified each paper based on the GitHub Marketplace categories. Mobile app categories have proved to be a representative label for identifying the core functionality of the products [33], [114]. Following that, we use categories to identify the core scope of functionality perceived by the app and action providers in our study. In the GitHub Marketplace, the product owner defined one or more categories for each app and action. Categories are a common practice in marketplaces that offer end users a convenient way to find products and services and possibly compare and choose among different alternatives. GitHub apps and actions can be classified into one or more categories.

We followed a protocol for the manual categorization of the papers. The first two co-authors of the paper each carefully reviewed 515 Paper abstracts and categorized them into up to three categories. For the first classification, we compared the results, and a total of 147 Disagreements were found in at least one of the categories. The two authors discussed these conflicts and could converge their opinion for 39 of these papers. In the case that no agreement could be made, the third co-author was involved. The third co-author was provided with the list of 108 Papers and independently categorized them. Then, the categorized results were reviewed by the authors, and the list was finalized.

As a result of this process, 53 Papers were identified by all the authors as irrelevant to the scope of this study and could not be mapped into any marketplace category. Hence, the answer to **RQ2** as detailed in Section 3.5 is based on 462

Papers. For identifying the degree of agreement among the annotators, we cross-annotated 80 Papers. The degree of Kappa's agreement among the annotators was for the *first* classification [111] and the agreement level was 0.77. For the *second* classification based on research type [113], the Kappa agreement was 0.82. We also calculated the Kappa level of agreement as $= 0.71$ for the *third* classification. The Kappa agreement above 0.6 is considered a substantial agreement [115]. It is worth noting that for the analysis of replicability status, the authors checked every paper for the replication material (data and tools) and reported the status. This was independent of the authors' subjective opinion, and hence no disagreement is expected.

3.3.3 Gap Analysis

We performed quantitative and qualitative comparisons to diagnose the gap between research and practice. We used the three schema from **RQ2** to perform our comparisons.

We were interested to understand and further compare with literature the *nature of core analytics* provided by the Marketplace tools [111]. To this end, we used a combination of NLP and manual processes. Buse and Zimmermann [111] schema identifies the nine different analytical tools as the intersection between the time stamp of the data (past, present, future) and the three insight types (exploration, analysis, and experimentation). We used `NLTK` library to identify the tense of the majority of the verbs in the app and action descriptions and formed a thesaurus of vocabulary for each of the three insight types using the main paper [111] and the `WordNet`. We tagged each paper in at most three categories. This automated approach was not definitive but rather made the manual analysis easier. We then manually went through the app and action descriptions and classified them into the appropriate category using our best judgment. As a result, we categorized each

tool at least on one of the nine categories introduced by Buse and Zimmermann [111].

Also, we compared the *population* of researchers active in automating a particular software process (category). We calculated the number of distinct paper authors and distinct app and action developers in our data of **RQ1** and **RQ2**. We identified distinct GitHub developers using their unique IDs. However, we found several different formats for author names in our mapping study (**RQ2**). To avoid counting duplicate authors, we performed `substring matching` to identify the different formats of the same name. Further, to distinguish between distinct authors with identical names, we performed a manual inspection of research profiles on the internet by comparing their affiliations and co-authors. We highlighted categories where a higher proportion of researchers are working compared to developers and vice versa. We further analyzed the proportion of active individuals in automating each category process relative to the number of published papers (researchers, **RQ2**) and products (developers, **RQ1**).

Further, we compared the *popularity* of studies with the products. As for the popularity in practice, we used the average number of installs (for apps), and the average number of stars (for actions) per each category as the proxy. For apps and actions, we got the average number of installs and the average number of stars per category by taking the total number of installs or stars for all apps or actions in a category and dividing it by the total number of apps in that category. For the academic popularity, we used GOOGLE SCHOLAR's API and calculated the average number of citations *per year* across all the papers mapped to each category. For each paper, we divided the number of citations by the number of years from the time they were published and calculated the average number of citations per category as described for apps and actions (above). We ranked each category based on these popularity proxies. Also, We compared co-occurrence and overlap between

categories among the marketplace tools (**RQ1**) and the mapped studies (**RQ2**).

We also compared the perspective of researchers and practitioners on the relevance and connection between different categories. New technologies and innovations appear from the synergy between topics. For example, Vasilescu et al. [116] discussed approaches such as “related-by-usage" (two languages used by the same authors in same projects) and “related-by-knowledge". Hence, we identified the most active and inactive intersection between the categories. In addition, we compared the content and description of marketplace tools with state-of-the-art studies. We used the short descriptions and full descriptions of apps and actions to extract the key topics in the marketplace (**RQ1**). In **RQ2** we extract these topics from the paper title and abstracts.

3.4 RQ1: Observations from Mining GitHub Marketplace

We introduce GitHub Marketplace and its data by **RQ1** as we are the first study on analyzing the GitHub Marketplace. To this end, we performed an **explorative study** on all the available data attributes (see Figure 3.2) and performed comparisons whenever possible. In contrast to the known app stores for the research community, such as Google Play, the GitHub Marketplace provides a moderate degree of customization for the developers. This results in non-unified information across apps and actions within the marketplace. For instance, only 5.01% of apps provide the information of their customers. In this study, we focused on the information that is available for all the apps and actions.

We scraped 440 Apps and 7,878 Actions. However, we could not retrieve the details of one app and 32 Actions due to HTTP status code errors. As a result, the following observations are based on an analysis of 439 Apps and 7,846 Actions. We discuss the results of our observations based on the characteristics of the platform,

developer, and user attributes separately (see Figure 3.2).

3.4.1 Developers Attributes

Developer attributes refer to the information provided by the developers at the time of publishing their products on the Marketplace. Names are unique identifiers for products (apps and actions) on the marketplace. The name is limited to 255 Characters. Products cannot have the same name as an existing GitHub account unless it is the developer's own user or organization name. For actions, the name should not match an existing marketplace category nor can it have the same name as a published action.

Categories

Each App and Action in the marketplace can be listed in one or two (primary and secondary) categories by its developer. Currently, 32 Categories exist in the marketplace. These tool categories and their definition as defined by the GitHub Marketplace are provided in Table 3.2. There are two categories exclusive to Apps and two are exclusive to Actions.

Table 3.2: Categories of apps and actions on the GitHub Marketplace.

ID	Type	Category	Category Description
C1	Both	API management	Structure your API infrastructure to enable various internet gateways to interact with your service.
C2	Both	Testing	Eliminate bugs and ship with more confidence by adding these tools to your workflow.
C3	Both	Utilities	Auxiliary tools to enhance your experience on GitHub.
C4	Both	Reporting	Get insights into how your teams are developing software using GitHub.
C5	Both	Continuous integration	Automatically build and test your code as you push it to GitHub, preventing bugs from being deployed to production.

Continued on next page

Table 3.2 – continued from previous page

ID	Type	Category	Category Description
C6	Both	Publishing	Get your site ready for production so you can get the word out.
C7	Both	Support	Get your team and customers the help they need.
C8	Both	Project management	Organize, manage, and track your project with tools that build on top of issues and pull requests.
C9	Both	Code review	Ensure your code meets quality standards and ship with confidence.
C10	Both	Deployment	Streamline your code deployment so you can focus on your product.
C11	Both	Chat	Bring GitHub into your conversations.
C12	Action	Community	Tools for the community.
C13	Both	Container CI	Continuous integration for container applications.
C14	Both	Dependency Mgmt.	Secure and manage your third-party dependencies.
C15	Both	AI Assisted	Tools that are superpowered with AI (artificial intelligence) to help you be a better developer.
C16	Both	Open Source Mgmt.	Running open source projects can be hard. Here are some tools to make that process a little more fun and a ton more manageable.
C17	Both	Security	Find, fix, and prevent security vulnerabilities before they can be exploited.
C18	Both	Monitoring	Monitor the impact of your code changes. Measure performance, track errors, and analyze your application.
C19	Both	Code quality	Automate your code review with style, quality, security, and test-coverage checks when you need them.
C20	Both	Localization	Extend your software's reach. Localize and translate continuously from GitHub.
C21	Both	Desktop tools	Developer tools that are run natively on your local machine.
C22	Both	Mobile	Improve your workflow for the small screen.
C23	Both	IDEs	Find the right interface to build, debug, and deploy your source code.

Continued on next page

Table 3.2 – continued from previous page

ID	Type	Category	Category Description
C24	Both	Mobile CI	Continuous integration for Mobile applications.
C25	Both	Code search	Query, index, or hash the semantics of your source code.
C26	Both	Code Scanning Ready	Static analysis, dynamic analysis, container scanning, linting, and fuzzing tools that integrate with GitHub Code Scanning SARIF Upload.
C27	Both	Learning	Get the skills you need to level up.
C28	Both	Time tracking	Track your progress, and predict how long a task will take based on your coding activity.
C29	Action	Game CI	Tools for building a CI pipeline for game development.
C30	Both	Backup Utils	Utilities providing periodic backups of your GitHub data.
C31	App	Content Att. API	Tools that use the Content Attachments API.
C32	App	GH Created	Tools created by the team at GitHub with love.

Figure 3.4 shows the number of Apps and Actions in each category. We have observed that Apps sometimes belong to more than two categories. 39 Apps have three categories and two Apps have four categories. For Apps, the categories 'Utilities' (118 Apps), 'Code Review' (81), and 'Project Management' (80) have the high-

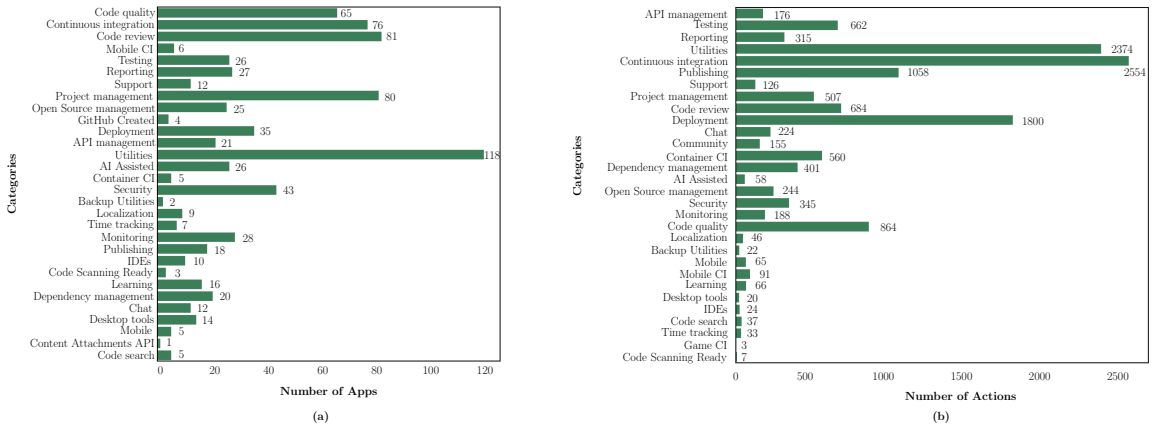


Figure 3.4: Each app or action is listed in one or more categories by its developer. (a) shows the categories and frequencies for apps, and (b) shows the categories and frequencies for actions.

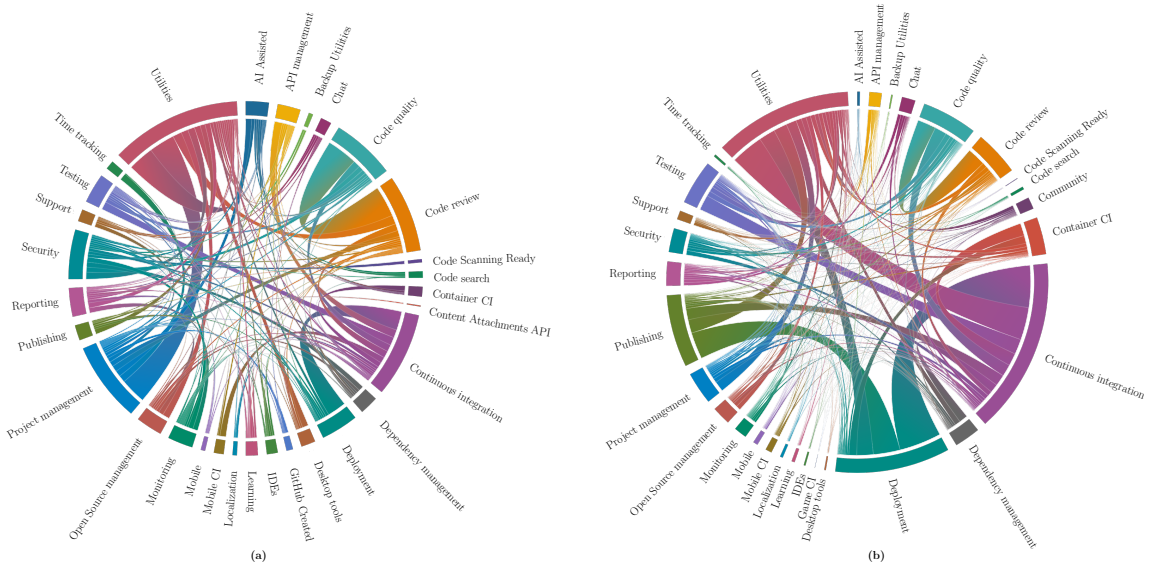


Figure 3.5: Frequency of categories appearing together for **(a)** apps and **(b)** actions on the marketplace.

est population. 'Continuous Integration' (2,554 Actions), 'Utilities' (2,374), and 'Deployment' (1,800) are the categories with the most number of Actions.

> 'Utilities' include the most number of Apps while 'Continuous Integration' has the most number of Actions in the marketplace.

We also investigate the frequency and magnitude of two categories appearing together. Figure 3.5 demonstrates the frequency of categories chosen together for Apps in Figure 3.5-(a) and Actions in Figure 3.5-(b). For Apps, the 'Code Quality' and 'Code Review' categories have the biggest intersection with 31 mutual Apps. This is followed by 'Utilities' and 'Project management' with 26 Apps. For Actions, 'Continuous integration' and 'Deployment' appear together for 603 Actions. The category 'Utilities' pairs with all other (28 out of 29 Categories) Action categories except 'Code Scanning Ready'. 'Game CI' (with three other categories), 'Code Scanning Ready' (4), and 'IDEs' (7) pair with the least number of other categories.

'Code Quality' and 'Code Review' have the highest overlap in terms of Apps. For Actions, 'Continuous integration' and 'Deployment' exhibit the highest intersection.

We use these categories for mapping studies in **RQ2**.

Short and full descriptions

GitHub allows product owners to add brief descriptions for their products in the marketplace to explain the product's main functionality. GitHub suggests keeping short descriptions at 40-80 Characters. 18.64% of the apps (82 out of 440) have short descriptions of less than 40 Characters while 33.64% of the apps (148 out of 440) have short descriptions that exceed 80 Characters. For Actions, 28.90% and 15.65% have a short description length of below 40 Characters and above 80 Characters, respectively. Figure 3.6 shows the distribution of the number of characters for apps and actions.

Each product has a dedicated page and URL on the marketplace which includes an elaborate description of the app or action. For apps, this full description should consist of two parts: a required 'Introductory description', and an optional 'Detailed description'[9].

[9]https://docs.GitHub.com/en/developers/GitHub-marketplace/listing-an-app-on-GitHub-marketplace/writing-a-listing-description-for-your-app#listing-details

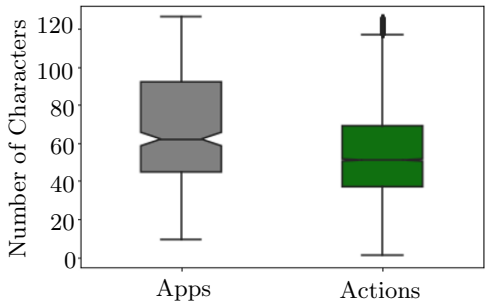


Figure 3.6: Number of characters in short descriptions for apps and actions on the GitHub Marketplace

Table 3.3: Top features in each marketplace category (extracted from the long description of products). The numbers in parenthesis stand for the frequencies of each feature appearing in the product descriptions.

Category	Top features
AI Assisted	label issue (21), unit test (12), dependency conflict (11), search code (4), code enhance (3)
API management	api specification (15), api documentation (13), swagger ui (13), performance test (5), schema change (3)
Backup Utilities	backup repository (5), data retention (4), daily backup (3), run workflow (3), cloud storage (2)
Chat	send message (110), slack notification (76), custom message (34), discord webhook (30), slack bot (29)
Code quality	code coverage (113), run test (82), static analysis (72), code review (27), automate code (5)
Code review	linting process (64), code coverage (59), automatically merge (6), test coverage (5), automate code (5)
Code scanning ready	infrastructure code (4), code scan (4), check vulnerability (4), automatically merge (2), automate deployment (2)
Code search	issue create (7), unit test (5), code enhance (3), best practice (3), search code (3)
Community	api key (25), push branch (20), create issue (17), open source (17), issue comment (14)
Container CI	container registry (126), run docker (82), build image (74), push image (61), machine learning (3)
Content Attachments API	real time (3), account login (2), embed issue (2), team collaboration (2), collaboration leanboard (2)
Continuous integration	docker image (347), run test (320), run workflow (206), docker container (160), unit test (36)
Dependency management	run test (54), project documentation (31), dependency conflict (11), fix vulnerability (7), update dependency (6)
Deployment	docker image (226), run build (136), cloud deploy (4), easy build (3), automatic deployment (3)
Desktop tools	cake script (12), UI test (5), desktop app (5), quick filter (5), issue tracker (2)

Continued on next page

Table 3.3 – continued from previous page

Category	Top features
Game CI	resource pack (11), optimize resource (5), build renpy (2), generate zip (2), file distribution (2)
GitHub Created	jira issue (4), friendly bot (3), learn skill (3), post comment (2), connect jira (2)
IDEs	code editor (4), smart IDE (3), code deploy (2), app inspection (2), sass solution (2)
Learning	update readme (8), execute workflow (7), smart IDE (3), learn skill (3), share knowledge (3)
Localization	check spelling (5), generate translation (4), machine translation (4), google translate (4), continuous localization (2)
Mobile	run test (26), android emulator (9), unit test (9), android app (8), deploy apps (3)
Mobile CI	run android (14), android emulator (8), android CI (7), upload artifact (6), deploy apps (3)
Monitoring	send notification (19), run lighthouse (16), html report (14), graphql inspector (5), crash reporting (3)
Open Source management	create workflow (31), close issue (25), label issue (29), post comment (5), merge pull-request (3)
Project management	create issue (68), label issue (59), create release (45), kanban board (9), track progress (7)
Publishing	create release (266), release note (178), docker image (132), build deploy (102), release tag (94)
Reporting	send message (52), unit test (50), slack notification (46), test report (33), see dashboard (2)
Security	security scan (53), scan repositories (51), security analysis (36), fix vulnerability (25), static analysis (21)
Support	support request (14), development support (13), label issue (10), powerful analytics (3), customer service (3)
Testing	unit test (103), test playbook (88), code coverage (66), test report (47), test automation (6)
Time tracking	todoist api (10), update readme (9), todo list (8), task organization (2), productivity growth (2)

Continued on next page

Table 3.3 – continued from previous page

Category	Top features
Utilities	project documentation (159), docker image (128), create workflow (108), dependency conflict (11), pull-request review (8)

To understand the features offered by the tools in each category, we mined long descriptions of all the tools in each category. Similar to the work of Al-Subaihin et al. [117] on mining features from mobile app descriptions, we refer to a feature as a claimed functionality mined from the product description. The most frequent feature of the products in the GitHub Marketplace is issue management. Categories 'Project management' (127 Mentions of this feature), 'Open Source management' (54), and 'Utilities' (53) all have products that offer a variety of this feature such as open or close issues, label issues, find an issue, or comment on an issue using a bot. Among these, `issue labeling` is the most prevalent feature. `Unit test` also appears frequently in products from the categories 'Testing' (103), and 'Reporting' (50). Also, apps and actions in the marketplace which facilitate `static analysis` mainly fall in 'Code quality' (72), 'and Security' (21).

The categories 'Dependency management' (17), 'AI Assisted' (11), 'Utilities' (11), and 'Open Source management' (6) all have products that help with `managing dependency`. These products mainly aid developers in updating dependencies, resolving dependency conflicts, creating dependency graphs, and documentation. A variety of products in overlapping categories also help developers with code coverage, API documentation, automating deployment, detecting and fixing vulnerabilities in software, and building and maintaining `Docker` images. Table 3.3 shows the top features from each category and the frequency in which they appear in product descriptions.

Developers provide long and short descriptions for their tools on the marketplace. Analysis of these texts for extracting software features shows a significant overlap between functionalities for different tool categories.

3.4.2 Platform Attributes

Al-Subaihin et al. [29] have explored the impact of platforms on software engineering practices. While the study is only limited to mobile app development, the impact of platforms on software is non-trivial. In this study, by 'platform attributes' we are referring to the attributes defined and provided by the GitHub Marketplace.

Developers

Overall, there are 5,928 Developers hosting their apps or actions in the marketplace. Among them, 368 published only apps, 5,509 published only actions, and 51 has both types of products on the platform. 20.70% (1,227) of the developers have more than one product (app or action) in the marketplace. *Azure* has the most number of products (one app and 35 Actions). This is also the highest number of actions by one provider in the GitHub Marketplace. This is followed by *Reviewdog* with 25 Actions (but with no apps) on the marketplace. *Devbotsxyz* owns the highest number of apps with five apps (but only one action).

Figure 3.7 shows the number of unique developers working on apps and actions in each marketplace category. 'Continuous Integration' (C5), 'Utilities' (C3), and 'Deployment' (C10) are the top three categories that have the highest number of active developers. This does not come as a surprise as these three categories also have the most number of apps and actions in the marketplace.On average, each developer has 1.40 Products (apps or actions) on the marketplace. For each category, if we consider the ratio between the number of developers to the number of products (i.e apps or actions), 'Continuous Integration' (C5, $ratio = 1.30$),

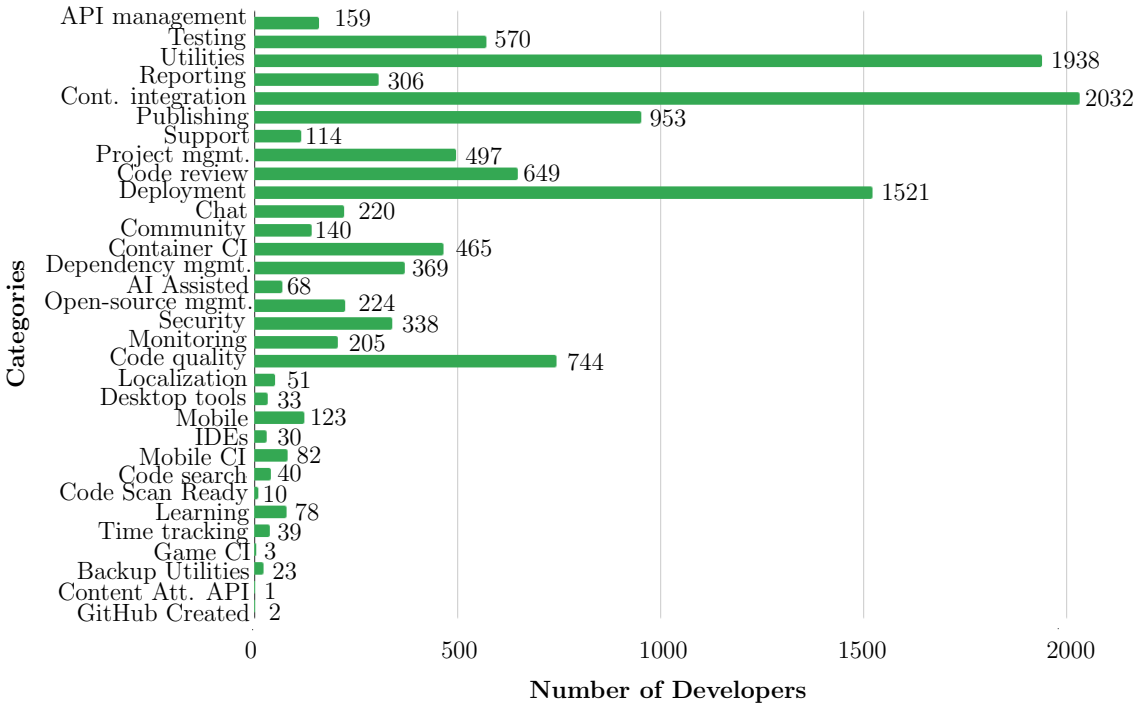


Figure 3.7: Number of developers (apps and actions) per category

'Utilities' (C3, $r = 1.29$), and 'Code Quality' (C19, $r = 1.25$) are ranked highest among all the categories. The very close ratios for all the categories imply that the majority of the developers (79.30%) in the marketplace tend to publish only one tool regardless of the category.

3.4.3 User and Usage Attributes

In the GitHub Marketplace, the information on users and usage is not consistent. In particular, the actions have information about the 'number of stars' while for apps, the 'number of installs' are provided. We will refer to the apps with the fewest number of installs as the least popular apps. Similarly, the least popular actions are the ones with the fewest number of stars. In the context of our study, while we use the number of stars and the number of installs as the most accessible user feedback, it is essential that the reader considers these limitations to this interpretation [35].

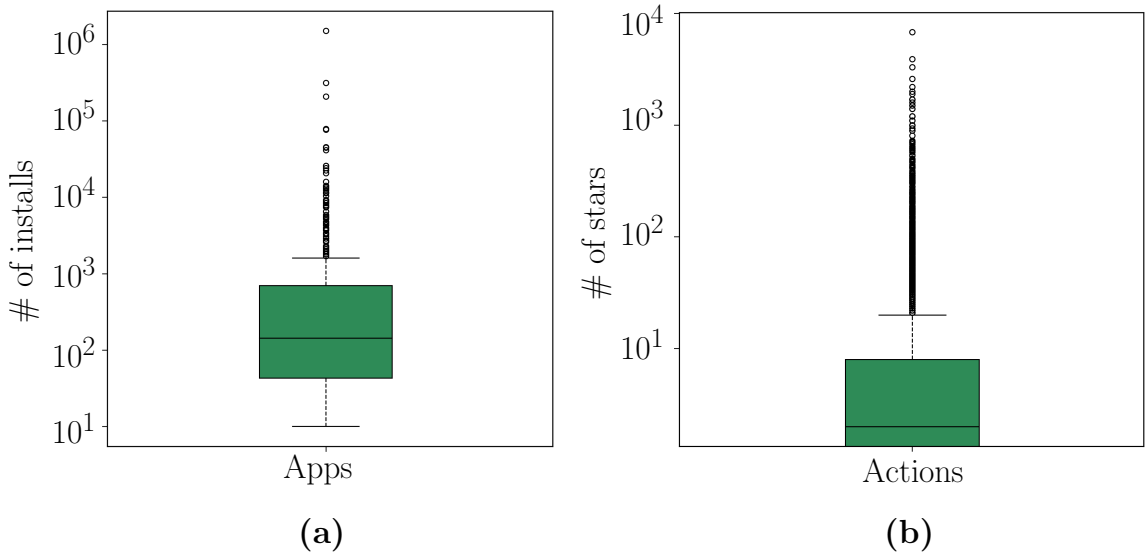


Figure 3.8: Distribution of number of installs for **(a)** apps and number of stars for **(b)** actions on the marketplace

Apps' Number of Installs

Due to GitHub's mechanism of masking the number of installs for the 'recommended apps', we only based our analysis in this section on 85% of the apps (374 apps out of 440) for which the number of installs is provided.

Figure 3.8-(a) is the Boxplot distribution of the 'number of installs' for apps. The highest number of installs was 1.5 Million which belonged to `GitHub Learning Lab` followed by `Travis CI` with 314,000 Installs. `Slack + GitHub` is the third most installed app with 207,000 Installs. Among the top quartile of most installed apps, three are developed by GitHub. On average, apps have been installed 7,530.12 times from the GitHub Marketplace. However, 92.25% of apps have less than the average number of installs. The median number of installs is only 143.5. Among the apps, the highest average number of installs belongs to the categories 'GitHub Created' (with 594,500 Installs), 'Learning' (115,748.08), and 'API Management' (75,320).

Actions' Number of Stars and Contributors

GitHub users star repositories to keep track of the projects they find interesting as well as to show appreciation to the repository maintainer for their efforts. GitHub may recommend related content to a user on their personal dashboard based on their starred repositories.

For this reason, users also star repositories in order to find similar projects on GitHub. Many of GitHub's repository rankings depend on the number of stars a repository has[10]. Similarly, in our analysis, we also use the number of stars as a popularity metric for the actions on the marketplace. `Super-Linter` developed by `GitHub` has the highest number of stars for actions with 6,800 Stars. This is followed by `yq - portable YAML processor` developed by `mikefarah` with 3,900 Stars. Figure 3.8-(b) demonstrates the distribution of the star ratings among actions. `Azure` has the most number of actions (35 Actions) followed by `Reviewdog` (25). When looking into the proportion of the number of actions in a category and the number of stars, 'Time tracking' and 'Community' are the top two categories while 'Game CI' and 'Backup Utilities' are the least popular.

Additionally, the marketplace lists all the contributors to an action. Contributors are the software developers who open an issue, propose a pull request, or commit any change to the default branch of an action repository. 7,111 Actions (90.63%) in the marketplace have at least one contributor. 61.81% Actions have only one contributor. On the other hand, a total of 145 Actions have 12 Contributors, which is the largest number of contributors for actions on GitHub.

[10]https : / / docs . GitHub . com / en / get - started / exploring - projects - on - GitHub / saving - repositories-with-stars

We measured the popularity of apps and actions based on the number of installs and the number of stars, respectively. The apps created by GitHub (C32) and the apps on 'Learning' (C27) are the top two app categories with the highest average number of installs. 'Time tracking' (C28) and 'Community' (C12) have the highest average number of stars for actions.

3.5 RQ2: Mapping Studies of Open Source Software Engineering Literature

In the previous section, we explored the status quo of the GitHub Marketplace, which are the tools practitioners use in the open source ecosystem (**RQ1**). To address the gap between academia and practice (**RQ3**), we performed a systematic mapping of software engineering studies in **RQ2**. In this mapping study, we used three schemas as described in Section 3.3.2.

First, we mapped the studies into the type of analytical tools introduced by Buse and Zimmermann [7], answering the question ***(i)*** *What are the types of analytical tools proposed in the context of OSS automation?*

Second, we mapped the studies according to their research and evaluation type as introduced by Wieringa et al. [44] and evaluated if the paper provides artifact (data, code) and their availability, answering the question ***(ii)*** *What is the research and evaluation type of papers?*

Third, we mapped the studies into the marketplace categories to perform a side-by-side comparison of the areas of automation, answering the question ***(iii)*** *What categories of marketplace tool is presented in the papers?*.

As discussed in Section 3.3.2, we gathered relevant papers on automation within the open-source software ecosystem between the years 2000 to 2021. We discuss the results of the mapping study in this section. Figure 3.9 shows the number

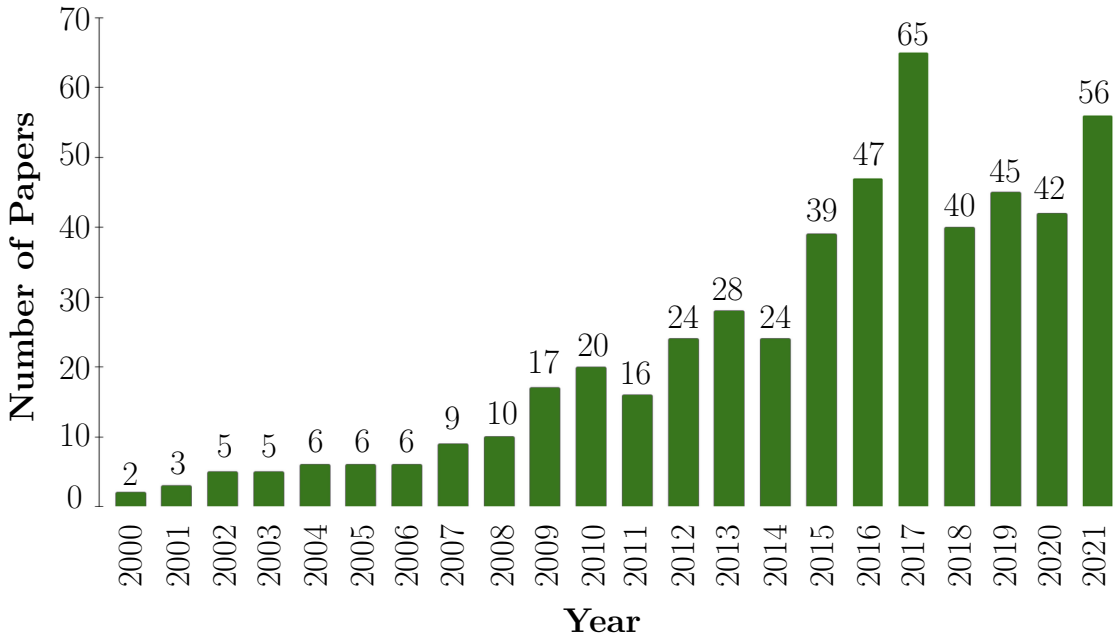


Figure 3.9: Number of categorized papers within our systematic mapping study (**RQ2**) in each year

of publications per year relevant to 'automation' and 'open source software engineering' in the top 20 Software engineering venues. There is a steep increase in the number of publications starting from the year 2009, and there is another peak in 2015 onward. 12.62% of the papers were published in 2017 followed by 10.87% published in 2021. There were 56 Publications in 2021, compared to only 16 Publications in 2011 (ten years period).

3.5.1 Mapping to the Types of Analytical Tools

Buse and Zimmermann [111] identified a set of analysis tools that translates well into the information needs of practitioners. These tools either find important conditions (exploration), explain the conditions in a diagnostic fashion (analysis), or compare the alternative conditions (experimentation). Each of these analyses can be carried out on the data from past, present, or target future predictions. Table 3.4 presents the distribution of papers across different categories of analytical tools. When looking into the results, 66.2% of the analysis done by the researchers was

Table 3.4: Number of mapped papers using the schema of analytical tools [111].

Tense	Insight	Tool Type	Num. of papers
Past	Exploration	Trends	38
Present	Exploration	Alerts	19
Future	Exploration	Forecasting	63
Past	Analysis	Summarization	49
Present	Analysis	Overlays	26
Future	Analysis	Goals	31
Past	Experimentation	Modeling	82
Present	Experimentation	Benchmarking	35
Future	Experimentation	Simulation	11

retrospective in nature, aimed at addressing the question of “what happened?". Moreover, 29.9% of the papers with automation tools discuss future predictions, and 33.4% of these studies work on the present information.

Notably, the category with the highest number of papers is ‘modeling’, which primarily involves retrospective experimentation to characterize normal development behavior and extract insights from past processes, accounting for approximately 32.0% of the total papers. Furthermore, the researchers exhibit a notable interest in ‘summarization’ tools, which involve capturing essential aspects of artifacts or groups of artifacts, comprising 21.0% of the total papers. Conversely, ‘simulation’, a type of analysis that aids in decision-making by evaluating different alternatives, is the least utilized category, accounting for only 4.1% of the mapped studies. Interestingly, a significant proportion (51.5%) of the studies undertake comparative analysis across alternative conditions. This approach serves various purposes, including learning from previous work through ‘modeling’, establishing best practices through ‘benchmarking’, or conducting what-if analysis via ‘simulation’. This highlights the researchers’ inclination towards exploring and

Table 3.5: Number of papers in the mapping study using the research type [113]

Research type	Num. of papers
Evaluation	269
Solution	204
Philosophical	47
Opinion	6
Experience	32

leveraging alternative conditions to gain insights and enhance their understanding of the subject matter.

A significant majority (66.2%) of academic studies primarily focus on conducting analytics on historical information. Conversely, experimentation for the possible future scenarios (simulation) and the analysis of the present state through interactive comparison (overlays) have the least popularity among the researchers, with 4.1% and 7.4% of the studies mapped into these categories.

3.5.2 Mapping of Research Type and Replication Status

The schema of research types by Wieringa et al. [113] introduces five different types of research. Evaluation research investigates the implementation of the problem or a technique in practice. We systematically mapped 269 Papers (58.23%) as 'evaluation' research. In comparison, 44.16% (204) papers are mapped into 'solutions'. Solutions are the papers that propose a solution along with its relevance, but the evaluation remains preliminary. We also found 47 'philosophical' papers (10.17%) which offer new conceptual frameworks and 32 'experience' papers (6.93%) with a focus on what happened without explaining why. Also, six papers described the opinion of the authors and we mapped them as 'opinion' papers. Table 3.5 summarizes these numbers.

To provide an overview of the studies based on the first two mapping schema

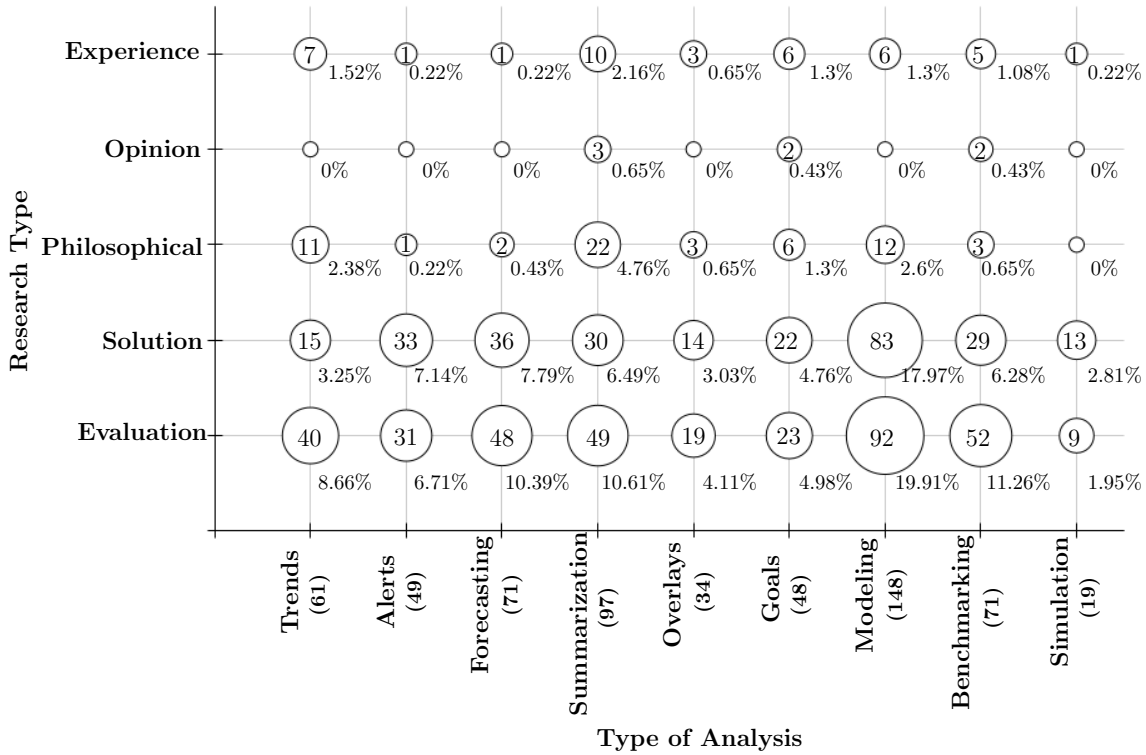


Figure 3.10: Papers in our mapping study: Research type and type of analysis. A paper can belong to multiple analysis and research types simultaneously.

(research type and analysis type), we present a two-dimensional bubble chart in Figure 3.10 where the size of the bubble is determined by the number of studies corresponding to the x-y coordinates. 92 Studies are categorized under evaluation research which uses modeling as their research technique to facilitate learning from previous work. This is followed by 'solution' research that uses 'modeling' with 83 mapping studies.

Evaluation research and solution proposal stand out as the dominant research types, comprising 58.23% and 44.16% of the studies, respectively. Upon analyzing the intersection between the analytical tool type and the research type, it became evident that the largest proportion (19.91%) of the studies falls within the intersection of 'modeling' and 'evaluation'. Additionally, the intersection of 'modeling' and 'solution' accounts for 17.97% of the studies, followed by 'benchmarking' and 'evaluation' (11.26%).

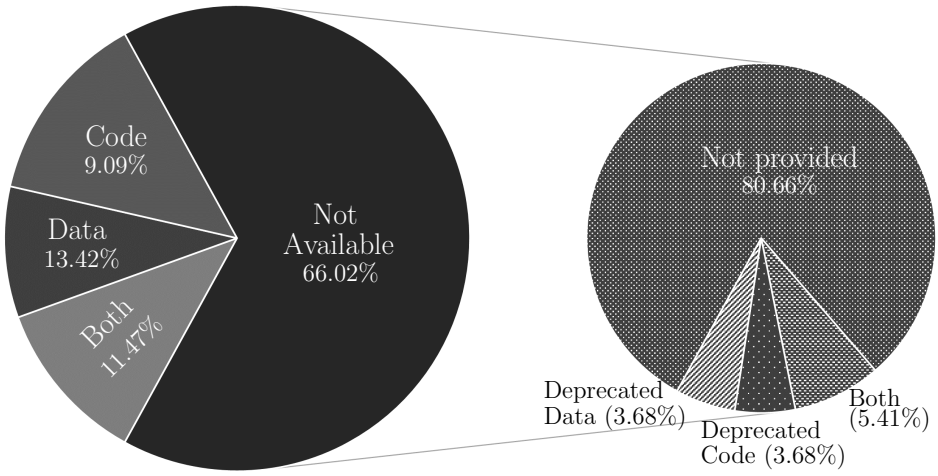


Figure 3.11: Replication Status of the papers in our systematic mapping study for automation tools in open source

We evaluated if each paper was accompanied by any tool or data package. Figure 3.11 shows that 66.02% (305) papers in our mapping study did not have any available artifacts. Out of these, 59 Studies include deprecated links to their artifacts. The remaining 246 Papers did not mention anything about the availability of their tools/data or include a link. Only 62 Papers (13.42%) shared their dataset publicly and even fewer papers (42) made their tool available (9.09%). Out of 462 Papers, only 53 Papers presented both their tool and dataset for the open-source research community.

66.02% of the papers failed to provide access to any code or data artifacts. This raises significant concerns regarding the reproducibility of the findings within software automation.

3.5.3 Mapping into GitHub Categories

Following the systemic protocol procedure described earlier (see Section 3.3.2), two authors independently mapped the papers included in the study into the app and

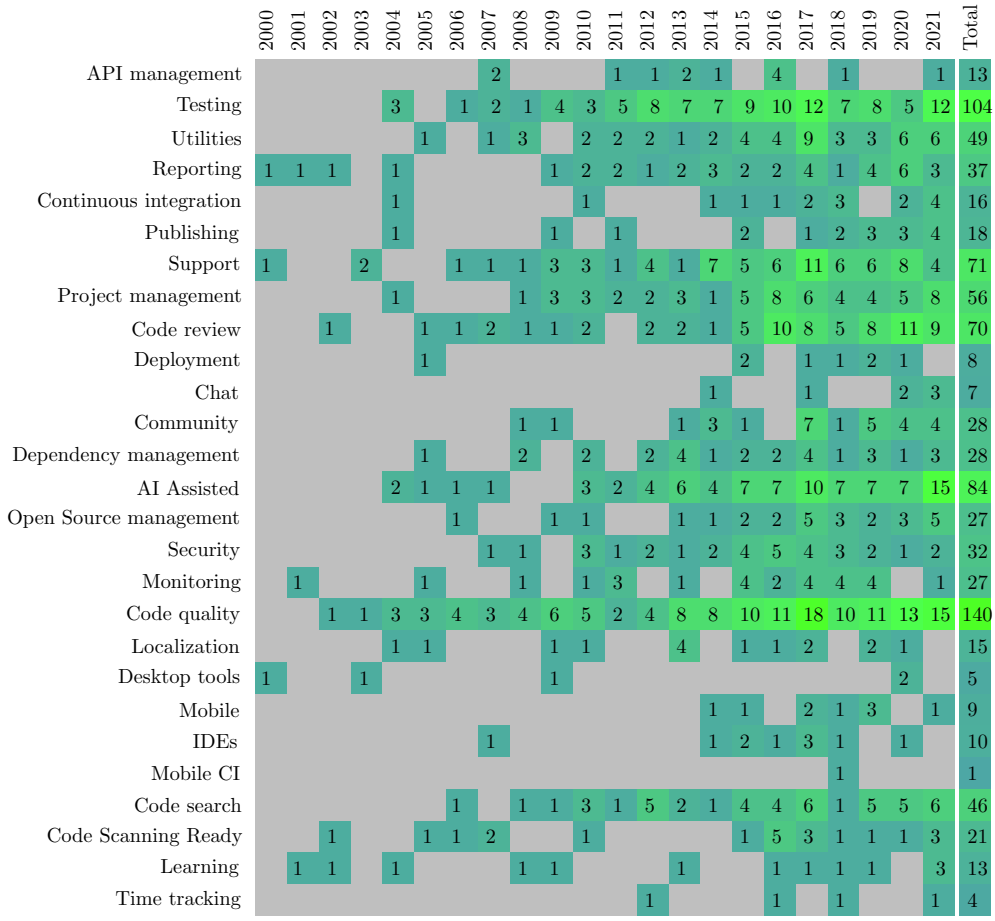


Figure 3.12: Number of papers in each year per category. Categories with zero number of papers were excluded from the heatmap.

action categories (32 Categories overall). In the mapping to the GitHub categories, we excluded the 'GitHub Created' as this category only applies to actions and not apps or papers. Also, the detailed mapping of each paper to each category is provided in Appendix A. We also examined how often two categories appeared together and how many papers the categories involved. Figure 3.12 shows the heatmap of the number of papers per year in each category.

The category of 'Code quality' stands out as the foremost area of focus, comprising 140 out of the total 462 papers. When we examine the distribution on a yearly

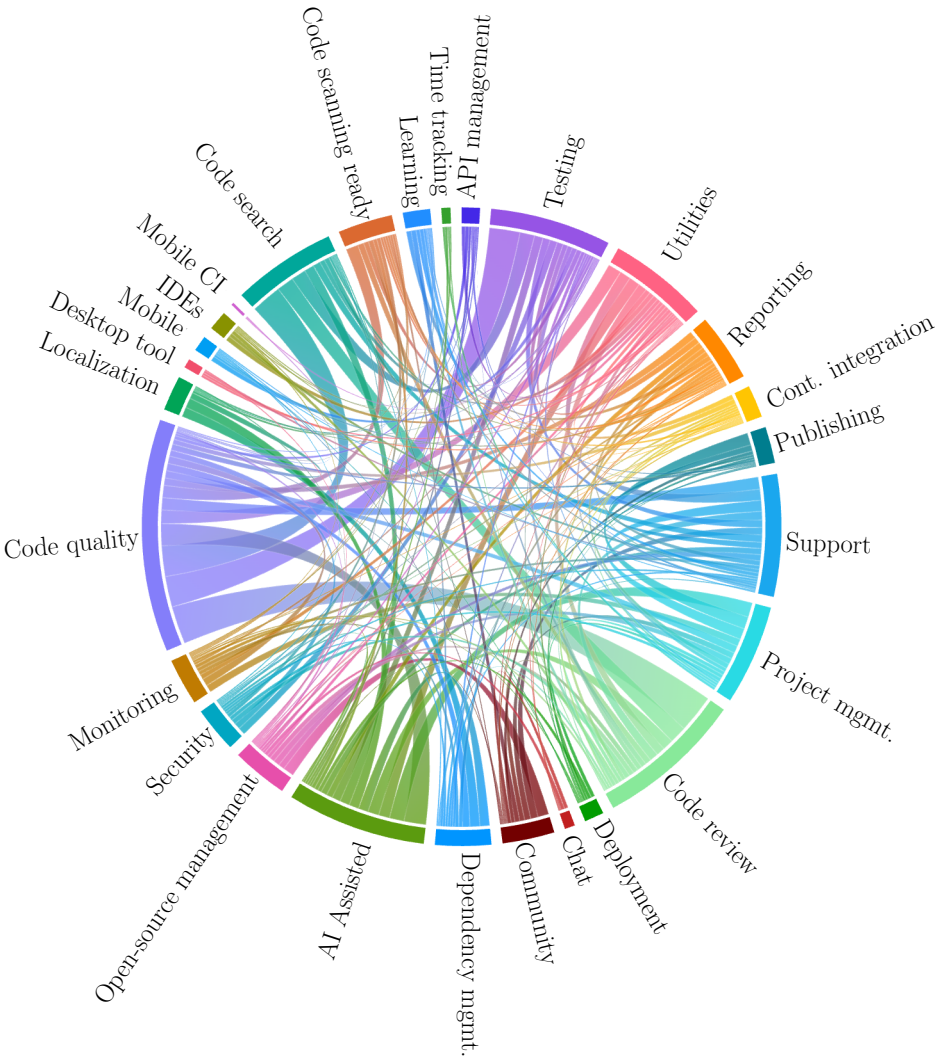


Figure 3.13: Frequency of categories appearing together for the papers in our analysis

basis, the highest number of papers dedicated to discussing automation tools is in 'Code quality' (18) for the year 2017, followed by the papers in 'Code quality' and 'AI Assisted' (15 each) in 2021. Figure 3.13 demonstrates the frequency of categories chosen together for papers. The biggest intersection belongs to 'Code quality' and 'Code review' with 31 Papers sharing these two categories. This is followed by 'Code quality' and 'Code search' with 25 Papers. 'Utilities' and 'Support' each pair with 22 out of 26 other categories for papers, followed by 'Code quality' (with 21 other categories). On the other hand, 'Mobile CI' pairs with only

two other categories while 'Desktop tool' and 'Chat' each intersect with four other categories, and 'Time tracking' appears together with five other categories.

'Code quality' emerges as the most extensively researched area among all the categories. Within this category, papers have a significant presence in related areas of 'Code review' (31 times) and 'Code search' (25 times), indicating their close association. in 'Code review' (31 times) and 'Code search' (25 times).

To understand the contents of these papers in further detail, we extracted bigrams from paper titles and abstracts for all the papers per category. We followed the same protocol and process as for extracting bigrams for tools on the marketplace (see Table 3.3). These bigrams represent the key features or phrases in each category.

In literature, there are several common features that appear across multiple categories. One such feature is `bug reporting`, which is found in 11 categories, namely: 'Project management' (46 mentions), 'AI Assisted' (45), 'Code quality' (30), 'Code review' (27), 'Reporting' (22), 'Support' (20), 'Code search' (18), and 'Monitoring' (12). Another widely shared feature is `requirement traceability`, which is the focus of papers in several categories. Specifically, 'Code review' (21), 'Utilities' (13), 'Dependency management' (11), 'IDEs' (10), 'AI Assisted' (10), and 'Localization' (8) all have papers discussing the topic of `requirement traceability`. Additionally, there are several categories that share papers facilitating `static analysis`. These categories include 'Code quality' (14), 'Code Scanning ready' (10), 'Utilities' (6), and 'Deployment' (4), all of which have papers related to this topic. Lastly, the topic of `test automation` is addressed in papers falling under the categories 'Testing' (17), 'Mobile' (4), 'Mobile CI' (4), and 'Open source management' (2). These common features and their presence across multiple categories highlight their significance and relevance in the literature.

Studies in different categories often discuss similar topics/features. The most common feature in literature is 'bug reporting' (appearing in 11 categories) followed by 'requirement traceability' (six categories), 'static analysis' (four categories), and 'test automation' (four categories).

We also looked into the number of distinct authors per category. We counted a total of 1,434 distinct authors for 462 Papers in our mapping study. Figure 3.14 shows the number of individual authors in each marketplace category. 'Code quality', 'Testing', and 'AI Assisted' are the top three categories that have the highest number of active researchers. These are also the top three categories with the most papers. Similarly, the category 'Mobile CI' has the least number of papers (one) as well as distinct authors (four). 25.54% of the papers (118 out of 462) have four authors which is also the median number of authors per paper. 16 Papers out of 462 (3.46%) have only one author.

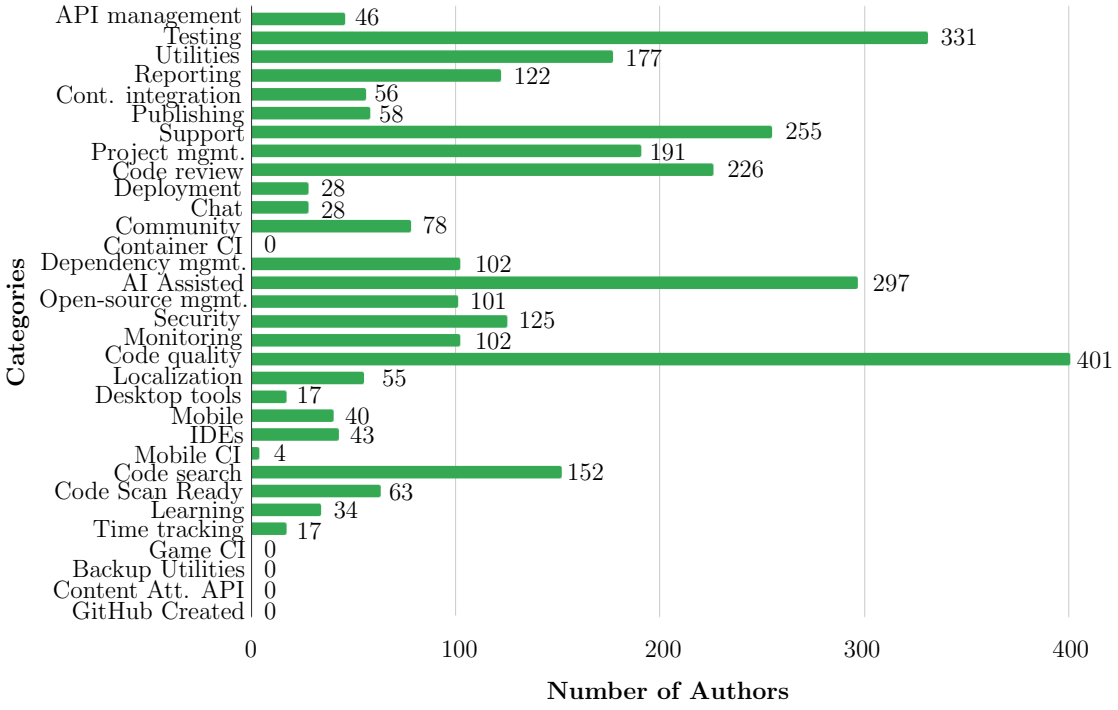


Figure 3.14: Number of authors per category

'Code quality' has the highest number of (401) distinct authors, Following closely behind are the categories 'Testing' (331) and 'AI Assisted' (297).

3.6 RQ3: The Gap

We performed a systematic study of the marketplace and the literature to perform a gap analysis in **RQ3**. We performed this comparison in three dimensions, (i) popularity of the automation tasks, (ii) type of analytical questions and tools in state of the art versus the practice, and (iii) features and technologies in these automation tools.

3.6.1 Comparison of Nature of the Analytical Tools

Multiple efforts were placed to address the "right question". Bagel and Zimmermann [118] performed a holistic survey with developers at Microsoft to understand the questions of interest in software teams and listed 145 Questions that could be of interest to the community. Nayebi et al. [90] evaluated the requirements from release engineers for mobile app analysis. These efforts are often employing surveys to understand developers' perceptions. Fernandez et al. [119] performed a large-scale survey with practitioners to identify the pain points of software requirements engineering. These are only a few samples of such surveys to identify the studies relevant to real-world practices. Among all, the study of Buse and Zimmermann [111] targeted the diverse information needed from stakeholders and provided a spectrum of analytical tools, their importance, and the difficulty of implementation.

The use of flat measurement does not provide deep insight. Layering several types of analysis often increases analytical decision-making. To move beyond the information toward insight, Buse and Zimemmrann [111] offered a spectrum of analytical tools for software development activities. They invested in the infor-

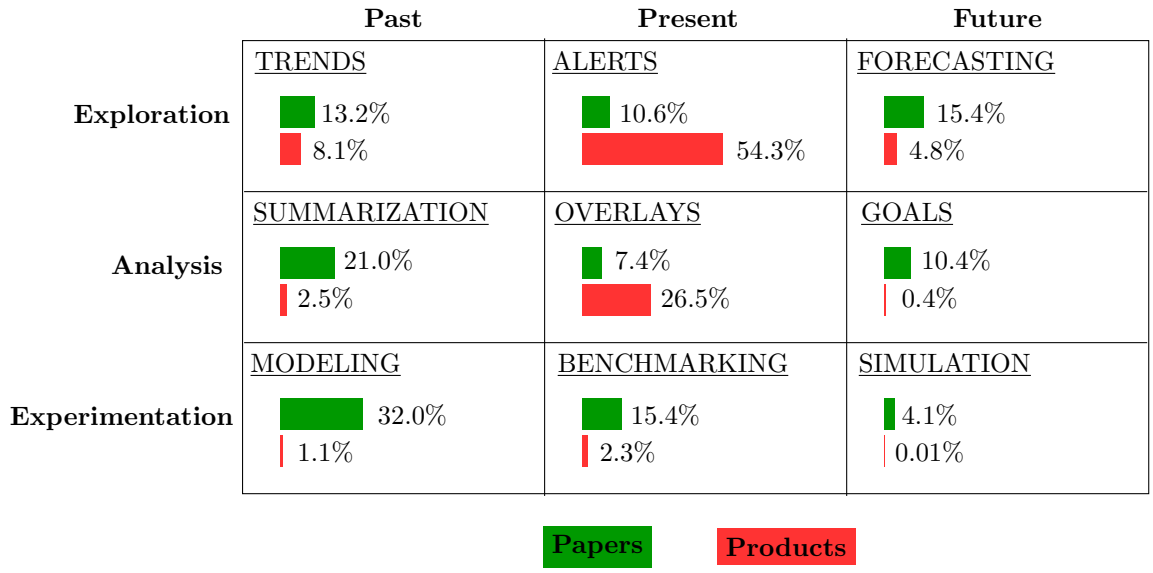


Figure 3.15: The number of marketplace products and the number of papers in the spectrum of analytic tools for development activities adopted from [111]. Each paper or tool can be categorized into multiple classes, and therefore the sum of percentages can exceed 100%.

mation needs of decision-makers in software engineering teams. According to the study of Buse and Zimmermann [111], the retrospective analysis of "How and why did it happen?" is the *most important* analytical question among practitioners. Their study showed answering "what will happen?" (in the future) is the most difficult for practitioners. We adopted the same spectrum of analytical tools and manually mapped the papers in our systematic mapping study into these categories (**RQ2**). To perform gap analysis, we similarly identified the type of GitHub Marketplace tools on the same analytical tool spectrum. Figure 3.15 demonstrates the percentage of papers and the percentage of marketplace tools in each of the nine categories following the schema of analytical tools [111].

Overall, experimentation has less tool support in practice. Modeling, Benchmarking, and Simulation all together have 3.41% of the tools of the marketplace. The least number of tools and paper both belong to the Simulation category. Simulations assist development teams with identifying the best and the worst-case scenarios and assist the managers with contingency planning. Simulation is also

among the more difficult analytical categories for tool development, according to Buse and Zimmermann [111]. Experimentation has the highest number of papers overall. 51.5% of the papers in our literature review for automation in the open-source ecosystem belong to experimentation. On the flip side, 64.9% of the marketplace tools target exploration (i.e., Trends, Alerts, and Forecasting) to identify important conditions in software projects. This compares with the 39.2% of the papers in this category. Analysis (i.e., Summarization, Overlays, and Goals) has 38.8% of the papers and 29.4% of the tools.

> Experimentation tools for Modeling, Benchmarking, and Simulation are the least popular among practitioners but has the highest proportion (51.5%) of the research papers. It appears that the majority of the tools (67.2%) in the marketplace are focused on exploration and finding important conditions in a repository (Trends, Alerts, and Forecasting).

Modeling demonstrated the most significant difference between research and practice activities. Modeling tools build a representation of the historical activities of a repository for the purpose of comparison. We could map 32.0% of the studies from our literature review to this category, while only 1.1% of the marketplace products are considered modeling tools. On the flip side, the marketplace automation tools are highly in support of Alerts. Alerts refer to the analytical tools that report anomalies in the course of the project life cycle. We could map 10.6% of the studies in our literature to this category, which shows there is a 43.7% percentage difference between our efforts in academia and practice.

> Researchers and practitioners demonstrate distinct emphases, particularly in the areas of 'Alerts' and 'Modeling' tools. The majority of marketplace tools (54.3%) are dedicated to 'Alerts', whereas researchers primarily concentrate on 'Modeling' tools, accounting for 32.0% of the research papers.

3.6.2 Tool Availability

The GitHub marketplace enlists tools that are available. Occasionally, some of these tools might be taken out of the marketplace by their providers. Hence, However, the marketplace involves multiple tools for automating the same task by different developers. This leads us to believe that for major workflows, alternative solutions are offered in the marketplace [120].

On the other hand, 66.02% of the papers discussing automation tools had no code or data associated with them. Among the rest, 20.56% provided some code associated with their study. 54.74% of these available tools were published after 2017 (the year the GitHub Marketplace was introduced). Among these available tools, 72.63% are available as a fully open source repository hosted on a third-party platform like GitHub (50.53% of them), BitBucket, or Sourcer. The rest are hosted on university domains or personal websites.

> Currently, only 20.56% of the automation tools developed by researchers for open-source ecosystems are available to the public. Almost half of these tools are hosted on personal domains. This is compared with a pool of available tools for automation governed by the marketplace.

3.6.3 The Popularity of Automation Tasks

We chose three approximations for measuring and comparing the popularity of particular information tasks between academia and practice;

First, we compared the number of publications with the number of tools offered in each of the categories (See Figure 3.16-b),

Second, we compared the number of GitHub tool developers with the researchers published in the tools offered in each category in GitHub to the number of papers (See Figure 3.16-a),

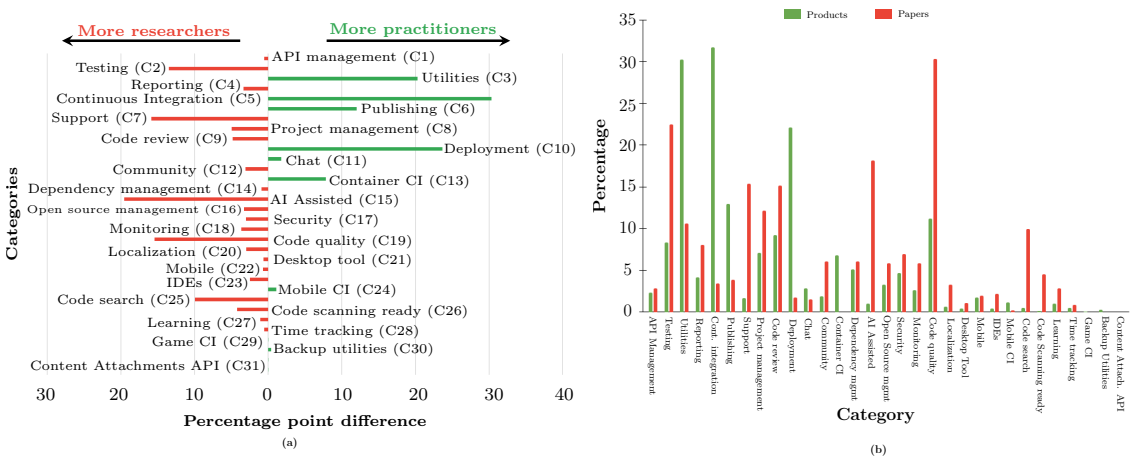


Figure 3.16: Difference in (a) the number of practitioners and the number of researchers (red shows a higher percentage of researchers working in a category, while green indicate a higher percentage of practitioners); and (b) the percentage of products (green) and papers (red) in each category.

Third, we compared the number of downloads and stars for GitHub tools with the number of citations of publications (See Figure 3.17).

First, we calculated the percentage of the papers and the marketplace products (apps and actions) in each category. The difference is less than one PERCENTAGE POINT (PP) for nine categories and more than 10 pp for seven other categories. The distribution of papers and marketplace products varies greatly across categories. For example, 'Continuous Integration' is the #1 and #4 category for actions and apps, respectively. However, this category contains only 16 (3.46%) papers and is ranked 17 out of 27 Categories with at least one paper. On the other hand, 22.51% (104) Papers in our mapping study fall under the 'Testing' category, while only 5.92% (26) Apps and 8.44% (662) Actions on the marketplace fall under this category. 10 Categories have a higher percentage of marketplace products. This difference is more than 15 pp for categories 'Continuous integration' (28.28 pp), 'Deployment' (20.42 pp), and 'Utilities' (19.64 pp).

In contrast, 'Code quality' (19.09 pp), 'AI assisted' (17.17 pp), 'Testing' (14.21 pp), 'Support' (13.7 pp), and 17 other categories have a higher percentage of mapped

papers into these categories compared to the percentage of marketplace products. There were four[11] categories on the GitHub Marketplace for which we did not find any papers within our systematic mapping study, namely 'Container CI', 'Game CI', 'Backup Utilities', and 'Content Attachments API'. Barring these four categories, 'Mobile', 'Time tracking', 'API Management', 'Desktop tools', 'Mobile CI', and 'Dependency management' had the least percentage point difference (ranging from 0.2 pp to 0.98 pp) between the proportion of the papers and the marketplace products. Figure 3.16(b) shows the percentage of the number of papers in the marketplace versus the systematic mapping.

There is a considerable difference in the distribution of papers and marketplace products across categories. 'Continuous Integration' and 'Code quality' are notable examples, with the former having a high percentage of marketplace products ($pp = 28.28$) and the latter having a high percentage of mapped papers ($pp = 19.09$).

Second, we looked into the distinct number of developers. There are 1,434 distinct authors and 5,937 distinct developers in our dataset of 462 Papers and 8,318 GitHub products (440 Apps and 7,878 Actions). We presented the number of developers (Figure 3.7) and the number of authors (Figure 3.14) per category in **RQ1** and **RQ2**, respectively. By comparing the two, we found that researchers and practitioners are interested in different subject categories. Figure 3.16 shows the percentage point difference between the contribution of researchers and practitioners for each category. For eight categories, there is more than 10 PERCENTAGE POINT (PP) difference between the percentage of researchers and practitioners.

20.71% (297 out of 1,434) Researchers are publishing in the 'AI Assisted' category. In contrast, only 1.15% (68) Developers are building apps and actions in this category (this implies a 19.56 difference in the Percentage Point (pp)). We have

[11] Excluding 'GitHub Created' which we do not consider for the gap analysis for aforementioned reason

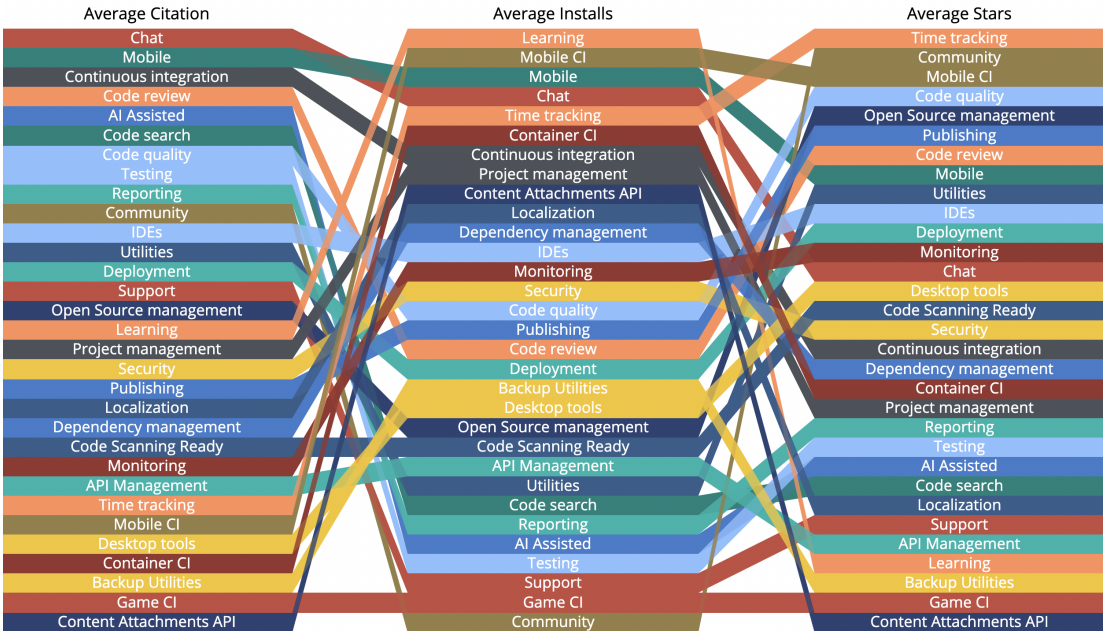


Figure 3.17: Category rankings based on three factors: the average number of paper citations per year (left), the average number of app installs (middle), and the average number of action stars (right).

similarly striking gaps between the percentage of authors and developers working in the 'Support' (15.86 pp more authors) and 'Code quality' (15.43 pp more authors) categories. 21 Categories out of 32 have a higher percentage (ranging from 0.53 pp to 19.56 pp higher) of authors compared to the percentage of developers working in those categories. On the other hand, only 3.91% (56) of the authors are publishing in the 'Continuous integration' category compared to 34.23% (2,032) of the developers contributing to the marketplace in this category. Glaring gaps such as this, where a higher percentage of developers are involved compared to researchers, also exist in other categories such as 'Deployment' (23.67 pp more developers) and 'Utilities' (20.3 pp more developers). Categories with the least difference between the proportion of researchers and developers are 'API Management' and 'Time tracking' (both having 0.53 pp more researchers), 'Desktop tools' (0.63 pp more researchers), 'Mobile' (0.72 pp more researchers), and 'Mobile CI' (1.1 pp more developers).

While developers are the providers and publishers of the actions, “contributors" can make changes to these open-source projects as well. Besides having a developer, most actions (90.63%) in the marketplace have one or more contributors. 64.49% of these contributors are working in the ‘Continuous integration’ category compared to 3.91% of the authors. Categories ‘Utilities’ (41.37 pp), ‘Deployment’ (39.75 pp), and ‘Publishing’ (19.75 pp) have considerably more percentages of contributors compared to authors. On the other hand, ‘AI Assisted’ (21.85 pp), ‘Support’ (17.49 pp), and ‘Code search’ (10.63 pp) have a higher percentage of authors than contributors on the marketplace.

In the open-source community, there exists a notable divergence between the research and practice domains. Researchers predominantly concentrate their efforts on enhancing ’Code quality,’ whereas practitioners exhibit a strong inclination towards ’Continuous integration.’ Remarkably, we observed a significant discrepancy of more than 10 percentage points in active participation between researchers and practitioners in a quarter of the categories (eight out of 32).

Third, we compared the usage and popularity of these topics. Figure 3.17 shows a bump chart that compares the rankings of categories based on the average number of paper citations, app installs, and action stars. The more popular categories in each column are shown on top while the least popular categories are on the bottom. The category ‘Mobile CI’ displays the most striking gap in popularity between researchers and practitioners. It is ranked 2^{nd} and 3^{rd} in terms of average installs and stars, respectively, while appearing at the 26^{th} position in the average citation ranking (one paper compared with 91 Actions). Time tracking’ is another category that is significantly less popular among researchers (#25) compared to practitioners (#1 among action users and #5 among the app users). ‘Container CI’, ‘Desktop tools’, ‘Monitoring’, and ‘Dependency management’ are other categories that are more favored by practitioners when compared to researchers. On the other

hand, categories 'Code review', 'AI Assisted', 'Code search', 'Testing', and 'Support' are more popular among researchers than both app and action users. The category 'Game CI' ranks #30 (out of 31) in all three metrics, making it one of the least popular categories both in research and in practice. 'Content Attachments API' (#31) and 'Backup Utilities' (#29) are also unpopular among researchers and action users although the categories rank 9^{th} and 19^{th}, respectively, in terms of the average number of installs.

> Among the popularly used categories in the GitHub Marketplace, 'Mobile CI' shows the biggest gap compared to popularity in use and referral with researchers. On the flip side, 'AI Assisted' papers are highly cited among researchers having the largest difference compared to the popularity of 'AI Assisted' tool usage in the Marketplace. 'Game CI' is equally the least used and referred to by both researchers and practitioners.

It is essential to note that there is an imbalance between the number of actions and papers in certain categories. 'Continuous integration', and 'Deployment' are two of the top three categories based on the number of developers. While there are 30 Action categories in the marketplace, GitHub actions were first introduced as an alternative to CI/CD services for GitHub repositories [105]. For this reason, it comes as no surprise that these categories are far more dominant in terms of the number of products. The other category, 'Utilities', also includes marketplace products related to CI/CD. These three categories also pair frequently between them for the products in the marketplace (see section 3.6.4). This as well results in more developers being involved in these categories.

3.6.4 Overlap Between Categories in Literature and in the Marketplace

The emergence of numerous technical innovations can be attributed to the powerful synergies that arise from the intersection of various models and techniques.

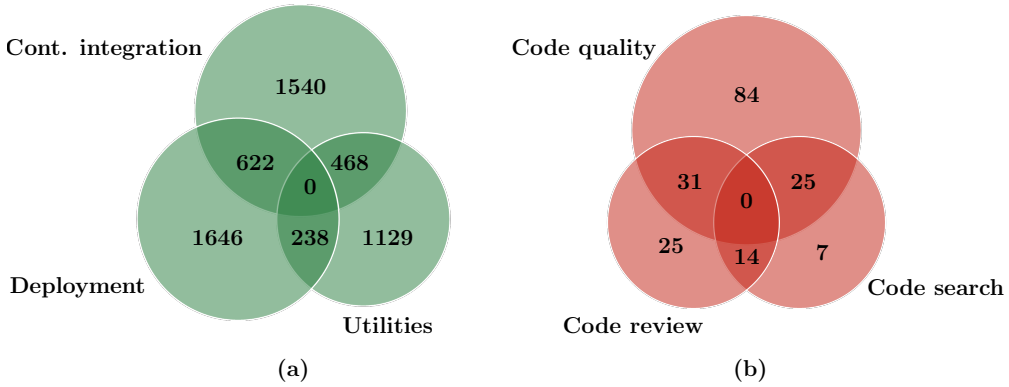


Figure 3.18: The intersection between categories is not equal in the state of the art and state of practice. The left Venn diagram (green) shows the largest overlaps between the marketplace categories (**RQ1**). The right (red) shows the biggest overlaps between the mapped studies (**RQ2**).

These intersectional synergies play a pivotal role in driving advancements and breakthroughs in both the marketplace and the field of open-source software engineering. As highlighted in Section 3.4.1, the marketplace allows for Apps and Actions to be listed under multiple categories. Similarly, we have categorized each paper into multiple categories. Our investigation focused on examining the intersections between these categories within the marketplace and the field of open-source software engineering literature. This analysis revealed that numerous technical innovations emerge from the synergistic combination of models and techniques.

Most frequently, 'Continuous integration' and 'Deployment' appear together in 7.51% (622 out of 8,285) Products in the marketplace, followed by 'Continuous integration' and 'Utilities' which are shared by 5.65% (468 out of 8,285) Products. However, in literature, we did not find any paper that falls under both 'Continuous integration' and 'Deployment'. On the other hand, 'Code quality' and 'Code review' are most frequently shared by studies in literature, appearing together in 6.71% (31 out of 462) Papers (see Figure 3.13). In contrast, there is no app in the marketplace that falls under both of these categories, and only 0.04% of ac-

tions share these two categories. Figure 3.18 shows the top three intersections of categories in the marketplace and in literature. The categories 'Continuous integration', 'Utilities', and 'Deployment' make up the top three pairs of categories shared among the marketplace products. These three categories also contain the most number of actions. Similarly, the three categories that feature in the most frequently shared categories in the literature- 'Code quality', 'Code review', and 'Code search' are ranked `#1`, `#7`, `#8`, respectively, based on the number of papers per category.

> The overlap between categories varies greatly in the marketplace and in literature. 'Continuous integration' and 'Deployment' has the largest intersection in the marketplace (appearing together 622 times) while never overlapping in literature. In contrast, 'Code quality' and 'Code review' overlap frequently in the literature (31 times) but scarcely in the marketplace (three times).

3.6.5 Qualitative Comparison of Automation Tools in Research and Practice

The marketplace products under the 'AI Assisted' category help with `code enhancement` (three mentions in product descriptions) while the studies mapped under this category discuss the task of `code generation` (5). Code enhancement tasks include improving code readability, fixing bugs in code, adding code comments, etc. On the other hand, code generation helps with producing situation-aware and business-aware applications or creating software for embedded systems, among other use cases. For 'API Management', both literature and marketplace products assist with `API documentation` and `API specification`. The marketplace offers products under 'Dependency management' that aid developers in finding and resolving `dependency conflict` (11) as well as `updating dependency` (6). Similarly, literature in this category proposes tools that help with `co-changing dependency` (4).

The features unit test and test automation are common among both literature and marketplace products that fall under the 'Testing' category. While 'Code Scanning ready' has products that facilitate code scanning (4) compared to papers that propose static analysis (10) and dynamic analysis (6) techniques, they both help with checking security-vulnerability. Products and papers under the 'Publishing' category help developers with their software releases. While the marketplace offers apps and actions that facilitate release creation (266), generating release-notes (178), and release tags (94), the literature discusses more efficient release scheduling (4). Both products and papers offer tools for cloud deployment. 'Code quality' also has products and papers that help with static analysis. As for the 'Code review' category, the marketplace has products for automatically merging pull requests (6), test coverage (5), code linting (64), and a variety of other use-cases. Literature, on the other hand, discuss code analysis (14), program comprehension (14), bug reporting (27), etc under this category.

For 'Community', the marketplace offers products for automatic issue creation and bot comments on issues. On the other hand, the literature discusses using issue reports (7) to identify duplicate issue reports or to label issues. For example, Di Sorbo et al. [121] analyzed over 6,000 Issue reports in 279 GitHub projects to investigate the nature of 'wontfix issues' and experimented with predicting 'wontfix issues' based on the title and description of issue reports. In the 'Learning' category, marketplace helps with learning skills and sharing knowledge, in comparison to literature which helps with knowledge acquisition (2). In the 'Monitoring' category, there are tools in the marketplace that help with crash reporting or sending notification. The mapped studies focus more on crash localization (3) and crash reproduction (5). For 'Security', the marketplace has tools for security analysis, and fixing vulnerability while the lit-

erature also discusses `code analysis` (5) and `detecting vulnerability` (14).

3.7 Discussion and Implications

To summarize, this paper involves three main contributions.

First, we performed the very first study on analyzing the GitHub Marketplace. The study of marketplaces has been of interest in the software engineering community since the emergence of mobile app stores. While there have been intense ongoing research and case studies in the domain of open-source software engineering, the marketplace, which is the platform for officially sharing automated tools for the open-source ecosystem, has never been studied. By analyzing the marketplace, we believe the research community can invest in mining this repository to gain more insight into different interactions and requirements within the open-source software community.

Second, we presented the results of a systematic mapping study of the literature in automation within the open-source community. We mapped the studies based on their research type, their type of analytical tool, and their subject category. There are multiple studies in the context of open source [122][123][124]. However, ours particularly provides a mapping to the perspective as seen officially by GitHub and the open-source developers and is the first to evaluate the types of analytical tools in this context.

Third, we performed a comprehensive comparison between our findings from mining the marketplace and performing the mapping study. This comparison of state-of-the-art and state-of-the-practice demonstrates and quantifies the difference between the research perspective and practitioners' perspective for automation in the open-source ecosystem.

A new software repository for mining - GitHub Marketplace is a software repository that *has not* been mined and presented in the software engineering research community. The marketplace is forming a two-sided platform (similar to the mobile app stores) where it provides visibility to both app developers and their users. The marketplace provides a combination of qualitative and quantitative data that can be of interest to many in the community. Most interestingly, in contrast to the mobile app stores, not only are the providers software developers but also the users are in software teams and are developers as well.

Relevance of research studies to practice - The community has known and acknowledged the gap between academia and industry. Lo et al. [125] gathered 571 Papers from ICSE and ESEC/FSE venues and asked practitioners to annotate their relevancy. The results showed that for 71% of all ratings, the publications were considered essential or worthwhile while emphasizing the room to improve the relevance and actionability of the research. This study later was replicated by French et al. [126] and showed similar trajectory. On the other hand, Devanbu et al. [127] invested and demonstrated that developers often hold strong a priori opinions about several technical aspects which are not well supported by evidence. Such beliefs formed by subjective understanding and personal experience are error-prone. In a follow-up article [128], they echoed the concern of many researchers, "Our life's work, embodied in research papers, counted so little toward forming the opinions of professional practitioners at one of the world's leading software companies?". Further, a more recent study by Shrikanth et al. [129] pointed to the disconnection between the beliefs of practitioners and what we achieve with empirical research. The authors advise practitioners and researchers to routinely evaluate their beliefs.

The community emphasizes performing relevant research for practitioners. The lack of access to data from industrial players has made the open-source community and projects the subject of many studies in the field. The research has been looking

into the open-source software hosted on GitHub. Tools such as GHTorrent[12] have been designed to facilitate research in the field. We have been hopeful that working on open source has at least gotten us closer to real-world issues. *While many of the studies start with a descriptive analysis of a sample of open source repositories, researchers have not directly evaluated the relevance of research in open source.* Our analysis of **RQ3** does indeed show the gap between the researchers' effort and the practitioners' work in the open-source community.

Mobilizing research results to practice - We believe that the marketplace has the potential to help knowledge mobilization if we focus our efforts in that direction. The analysis and mining of the GitHub Marketplace (such as the one done in our study) can *provide researchers with access to a diverse set of perceptions*. A deep look into the developers' ways of automating their tasks on repositories and observing their actions can further reduce the bias of questioning and increase our leverage in narrowing the gap. The *conference tool tracks* can encourage authors who are researching in open source to provide their tools and techniques as an action within the GitHub Marketplace. We think this is a straightforward and compatible way to make a real-world impact and move above silos of research, each performing a case study on a random set of open-source repositories. Further, actions being released on the marketplace can also address the issue with replicability and reproducibility of software engineering research.

In recent years, there have been centralized efforts along with several initiatives to encourage the reproducibility of research results in the community. Conferences offer tool tracks, and ACM[13] systematically evaluates and offers reproduction badges. GitHub Marketplace could be a sustainable alternative to releasing and maintaining tools. We argue that the marketplace provides a unique chance to get further exposure to the practitioners' world and ensure the relevance of the

[12]https://ghtorrent.org
[13]https://www.acm.org/publications/policies/artifact-review-badging

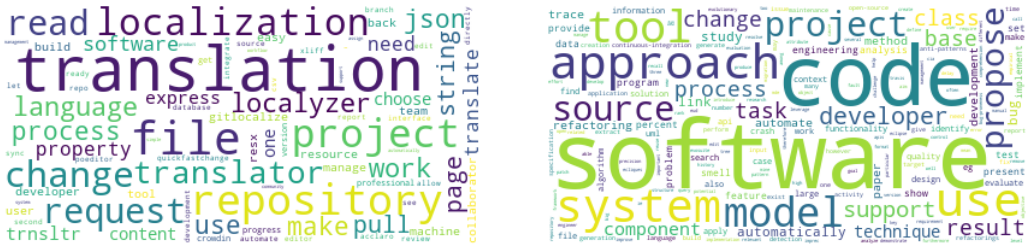
tools we are offering in academia. We suggest that whenever applicable[14], the conferences and the tool tracks encourage authors to publish their open source tools in the GitHub Marketplace. This is a proactive step toward reducing the knowledge mobilization gap and validating the automation tools in real-world scenarios.

3.8 Threats to Validity

This study attempts to obtain a picture of the current state of research and practice in the software engineering community. We have designed measures to elaborate on the state of the art and practice and the mobilization gap. These measures, however, have some threats to validity that we discuss below. We refer to the validity types introduced by Wohlin et al. [62] to structure these limitations.

As for the *conclusion validity*, the threat of drawing wrong conclusions for **RQ1** and **RQ2** are low as the nature of questions is descriptive. We used conventional statistical testing and visualizations to summarize our observations. We used categories as the main unit for our side-by-side comparison. As we used the marketplace categories and their definitions for our mapping study, we expect these mappings do not pose a huge validity threat. However, the area of categorization is fuzzy and one contribution (either the papers or apps and actions) can be categorized in multiple areas. To provide the readers with further insight, we provided an analysis of the intersection between the categories in Section 3.6.4. We used ratings and the number of stars to analyze the popularity of apps and actions in the marketplace (**RQ1**). The inaccuracy of these indicators has been discussed by Ruiz et al. [35] for mobile apps. Also, Borges et al. [88] warned about relying on the number of stars as a metric for repository quality- stating that more stars could mean effective marketing and advertising strategies and not necessarily solid software engineering principles and practices. These proxies and comparisons might

[14]Note that not all the automation tools are designed to automate GitHub workflows



(a) (b)

Figure 3.19: Word clouds for the categories **(a)** Localization and **(b)** Utilities generated using full descriptions of apps in the categories

pose a threat to the validity of the conclusions we have drawn.

When it comes to our diagnostic evaluation of the gap between academia and the industry (**RQ3**), there are a number of potential threats. We used the number of papers and authors, as well as the number of apps, actions, and their developers as a proxy for the extent of interest from the community. However, these measures might not be an accurate proxy. Moreover, as time passes, the number of citations, in particular, can be affected extensively. Similarly, the popularity (or the perceived value by the user) is inaccurate and only acts as a proxy. To mitigate the risk, we used the average number of citations among all the papers and the average number of installs and stars for apps and actions within each category.

As per the *construct validity*, the exclusion criteria applied to the systematic mapping may have caused us to ignore some relevant papers in the field. For instance, we collected the papers for our mapping study from the top 20 Venues listed in Google Scholar's ranking. This may have excluded some relevant papers from our study. Yet, we believe the search was extensive enough to cover important aspects for our comparison purpose. We also used the categories on the GitHub Marketplace to categorize the 515 Papers from our dataset. However, the academic and industry terminologies are not consistent. For instance, in the marketplace, 'Localization' is to 'Extend your software's reach. Localize and translate continuously

from GitHub'[15]. While in academia, localization often refers to bug localization. Thus, using just the category name to categorize the research papers and then mapping them to apps and actions on the marketplace could be error-prone. We also rely on the developers to choose appropriate categories for their products on GitHub. However, there could be developers having different perceptions of the categories, making this subjective to the developers' perception. To mitigate this risk, before mapping the studies into the categories in **RQ2**, all the annotators familiarized themselves with the official category definitions in the marketplace. We looked into samples of apps and actions within a category and discussed within the group to have similar understandings of the category definitions.

Further, we used NLP techniques on the app and action descriptions to generate word clouds for each category (a sample of word clouds for the 'Localization' and 'Utilities' categories are shown in Figure 3.19). While this was not good enough to allow us to automate the categorization process, it gave us a better idea of what the categories meant in the context of the GitHub Marketplace. For the final categorization, two authors of the paper went through all the products manually and finalized the categorization. This process could be biased, but the level of disagreement was within an acceptable range.

When it comes to the *internal validity*, in **RQ2**, we categorized the papers manually. Although the process was independent, after it was done, we discussed the disagreements and used a mediator to solicit an external opinion and adjusted the categories. However, there could still be bias and misunderstandings that threaten the internal validity. It would be ideal to find ways to automate the categorization process, which would make this work free from unintentional research bias. However, as we followed the established protocols of the empirical study and systematic mappings, we expect these biases are minimal.

[15]https://GitHub.com/marketplace?category=localization&query=&type=&verification=

We also used a combination of an automatic and a manual process to identify individual researchers. We used a combination of names, surnames, and affiliations to identify the individual authors in the field. We followed the process by manually inspecting scholar profiles and LinkedIn accounts. Yet, there is a small chance that some individuals were not identified properly and a few redundancies exist.

As per the *external validity*, for the academic studies (**RQ2**), the publication data and citations over time were available. However, the marketplace does not provide dates for apps and actions over time (neither in terms of the publication time nor their popularity status over time). As a result, we could not perform a time-based similar analysis or any analysis about the evolution of categories over time. This lack of historical data makes it hard to predict upcoming trends in the field. Hence, it is hard to comment on the generalization of the study beyond the scope of this empirical investigation. We expect that in the long run, new categories would be introduced or eliminated from the marketplace.

3.9 Conclusion

GitHub Marketplace is a software repository that provides tools for automation based on GitHub events. The marketplace provides a plethora of information about the tools available on this platform. These include developers, platforms, and users' usage data on the tools. We mined and analyzed GitHub Marketplace's provider, user, and platform attributes both quantitatively and qualitatively. We also presented systematic mapping study research results on automation tools for open-source software. For the 515 Papers in our review, we mapped them based on research and analytical tool types as well as their category of tasks. We quantified and highlighted the gaps helping researchers and practitioners to identify opportunities and synergies to work toward reducing the gap.

We found that although some of the automation topics in literature are widely used in practice, they have yet to align with the state of practice. More importantly, the researchers focused their automation efforts on experimentation, particularly comparing alternative conditions on historical data (i.e., Modeling). On the flip side, the practitioners are mostly developing tools for exploration and finding important conditions. In particular, Alerts, which consist of reporting unusual changes in the artifacts, have been the most popular. Further, we observed that in the marketplace, the percentage of automation tools relevant to 'Code quality' is less than one-fifth of the papers published in this field. We found that there is significant research work on topics such as 'Testing', 'Code quality', and 'Continuous integration' while practitioners are still not making full use of them. By introducing GitHub Marketplace as a new repository, we enable the mining community to take further steps in understanding and analyzing different automation tools, development, user, and usage behaviors. The summary of the state of the art helps the researchers to build on top of the existing work while practitioners can navigate the technology and find the synergies. We hope this leads to more vigorous knowledge mobilization within software teams.

Chapter 4

Conclusion and Future Work

The emergence of GitHub Actions allows developers to automate various tasks in a software development life cycle, from building and testing to deploying and releasing code. However, while GitHub Actions can simplify the development process, as we have found out in our previous research [120], developers often face challenges when working with Actions. One significant issue is limited functionality - developers may find that existing Actions do not perform all the tasks they need, requiring them to create new Actions from scratch. This can be time-consuming and may result in redundant and incomplete workflows. Another challenge is inadequate documentation and lack of support surrounding Action configuration, which can make it difficult for developers to understand how to use existing Actions effectively. In some cases, developers may not be familiar with the syntax and configuration required for YAML files, further complicating the process. This can lead to mistakes, bugs, and other issues that can slow down the development process.

4.1 What's Next?

Future research can address these challenges by conducting an analysis of similar Actions, identifying additional features, and recommending alternative Actions. This analysis will help reduce redundancy and promote completeness in work-

flow automation. Comparing similar Actions and providing recommendations for alternative Actions will assist developers in finding and reusing existing Actions instead of creating new ones from scratch. Furthermore, identifying additional features will help developers enhance their existing workflows and reduce the need for multiple Actions to perform the same tasks. Future research may also focus on assisting developers in configuring YAML files, helping developers overcome some of the challenges they face when working with Actions and reducing the time and effort required to complete development tasks. By optimizing workflows, saving time, and enhancing overall efficiency, future research can greatly benefit the field of software development.

The studies of this thesis present several other opportunities for future research. It is evident that GitHub Actions are transforming the nature of bots on GitHub. The adoption of these Actions into different processes can help with analyzing team practices and automating the workflows for increased developer productivity. Future research can focus on understanding the best practices, patterns, and anti-patterns by studying Apps and Actions in the OSS ecosystem. Additionally, research efforts can be directed towards designing better Apps and Actions that can benefit more teams, similar to the initiatives undertaken in the app store [130].

Our research study in [120] revealed that a significant number of developers (60.87%) find the creation of YAML files for GitHub Actions difficult and prone to errors. To address this issue and make GitHub Actions more accessible to a wider audience, future research can help to develop an automatic workflow generator framework. This framework will help simplify the process of creating YAML files for GitHub Actions, reducing the learning curve and potential for errors. This will also empower developers to create reliable and efficient workflows for their projects, without having to spend excessive time and effort on configuration. Last but not least, the biggest challenge for Action providers has been the *testing abil-*

ity of Actions as 25% of the developers mentioned they faced difficulty to test and debug their Actions [120]. Developing standardized testing frameworks and guidelines specifically tailored for GitHub Actions can help Action providers ensure the reliability and functionality of their Actions. Future research can explore the design and implementation of such frameworks and identify best practices for testing Actions effectively, such as analyzing logs, providing comprehensive error messages, and offering debugging utilities. This will help Action providers in identifying and resolving issues more efficiently.

4.2 Wrapping Up

In today's fast-paced world of software development, there are always new tools and technologies emerging. Among these, GitHub Actions, bolstered by the GitHub Marketplace, is a game-changing resource that has transformed the way developers work. Through our research, we have uncovered both the strengths and weaknesses of these tools, providing insights for practitioners and researchers. For example, we have found that the average number of stars for Actions *with* verified creators is 115, which is considerably more than 17.35, the average number of stars for Actions *without* a verified creator. To make their Actions more popular, we recommend that Action providers become verified creators, publish their Actions to the marketplace, and write comprehensive documentation for their Actions [120].

Our findings show that while practitioners heavily rely on automation tools for continuous integration and utilities, academics focus on code quality and testing. Moreover, although academic research has produced several automation tools over the years, only a fraction of those tools are available for practical use. This underscores the need for more collaboration and knowledge-sharing between researchers and developers to ensure that the latest advancements in automation are implemented in practice.

Furthermore, our survey of GitHub Actions users and developers highlights the importance of documentation, support, and security in the adoption of these tools. By addressing these concerns, we can promote greater visibility, re-usability, and effectiveness of GitHub Actions, while reducing redundancy and promoting completeness in workflow automation. Overall, our research provides valuable insights into the world of open-source software development, helping to drive innovation and collaboration in this exciting and dynamic field.

As we move forward, we must continue to bridge the gap between industry and academia, promoting knowledge sharing and collaboration to drive progress in software development. By building on the existing work and leveraging the insights gained from our research, we can pave the way for a more streamlined, efficient, and effective software development process, powered by the cutting-edge technology available on the GitHub Marketplace.

Bibliography

[1] G. F. Hoffnagle and W. E. Beregi, "Automating the software development process," *IBM Systems Journal*, vol. 24, no. 2, pp. 102–120, 1985. DOI: 10.1147/sj.242.0102.

[2] R. Hat, *Understanding automation*, https://www.redhat.com/en/topics/automation, [Accessed Feb 6, 2023], May 2022.

[3] D. S. N, S. D. S, D. Vijayasree, N. S. Roopa, and A. Arun, *A review on the process of automated software testing*, 2022. arXiv: 2209.03069 [cs.SE].

[4] X. Liu, "Process oriented analysis for software automation," in *2009 First International Workshop on Education Technology and Computer Science*, vol. 1, 2009, pp. 1131–1133. DOI: 10.1109/ETCS.2009.258.

[5] E. J. Barry, C. F. Kemerer, and S. A. Slaughter, "How software process automation affects software evolution: A longitudinal empirical analysis," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 1, pp. 1–31, 2007.

[6] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *2012 7th International Workshop on Automation of Software Test (AST)*, 2012, pp. 36–42. DOI: 10.1109/IWAST.2012.6228988.

[7] D. Ståhl and J. Bosch, "Experienced benefits of continuous integration in industry software product development: A case study," in *The 12th iasted international conference on software engineering,(innsbruck, austria, 2013)*, 2013, pp. 736–743.

[8] E. Soares, G. Sizilio, J. Santos, D. A. da Costa, and U. Kulesza, "The effects of continuous integration on software development: A systematic literature review," *Empirical Software Engineering*, vol. 27, no. 3, p. 78, 2022.

[9] K. K. Ng, C.-H. Chen, C. Lee, J. (Jiao, and Z.-X. Yang, "A systematic literature review on intelligent automation: Aligning concepts from theory, practice, and future perspectives," *Advanced Engineering Informatics*, vol. 47, p. 101 246, 2021, ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2021.101246. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S147403462100001X.

[10] N. Bleiel, "Collaborating in github," in *2016 IEEE International Professional Communication Conference (IPCC)*, 2016, pp. 1–3. DOI: 10.1109/IPCC.2016.7740497.

[11] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining github for research and education: Challenges and opportunities," *International Journal of Web Information Systems*, vol. 16, no. 4, pp. 451–473, 2020.

[12] IBM, *Open source*, https://www.ibm.com/topics/open-source, Accessed Feb 6, 2023.

[13] L. Erlenhov, F. G. d. O. Neto, and P. Leitner, "An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective," ser. ESEC/FSE 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 445–455, ISBN: 9781450370431. DOI: 10.1145/3368089.3409680. [Online]. Available: https://doi.org/10.1145/3368089.3409680.

[14] X. Wu, A. Gao, Y. Zhang, T. Wang, and Y. Tang, "A preliminary study of bots usage in open source community," in *Proceedings of the 13th Asia-Pacific Symposium on Internetware*, ser. Internetware '22, Hohhot, China: Association for Computing Machinery, 2022, pp. 175–180, ISBN: 9781450397803. DOI: 10.1145/3545258.3545284. [Online]. Available: https://doi.org/10.1145/3545258.3545284.

[15] M. Wyrich and J. Bogner, "Towards an autonomous bot for automatic source code refactoring," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, 2019, pp. 24–28. DOI: 10.1109/BotSE.2019.00015.

[16] M. Monperrus, "Explainable software bot contributions: Case study of automated bug fixes," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, IEEE, May 2019. DOI: 10.1109/botse.2019.00010. [Online]. Available: https://doi.org/10.1109%2Fbotse.2019.00010.

[17] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 84–94. DOI: 10.1109/ASE.2017.8115621.

[18] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 928–931, ISBN: 9781450342186. DOI: 10.1145/2950290.2983989. [Online]. Available: https://doi.org/10.1145/2950290.2983989.

[19] M. Wessel, B. M. de Souza, I. Steinmacher, *et al.*, "The power of bots: Characterizing and understanding bots in oss projects," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Nov. 2018. DOI: 10.1145/3274451. [Online]. Available: https://doi.org/10.1145/3274451.

[20] Z. Hu and E. Gehringer, "Use bots to improve github pull-request feedback," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19, Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 1262–1263, ISBN: 9781450358903. DOI: 10.1145/3287324.3293787. [Online]. Available: https://doi.org/10.1145/3287324.3293787.

[21] M. Campbell, *How sentry moved to github apps*, https://github.blog/2018-10-11-sentry-guest-post/, Accessed: May 15, 2023, Oct. 2018.

[22] A. Hermann, *Migrating to github apps, code climate shares their story*, https://github.blog/2018-09-25-migrating-to-github-apps-code-climate-shares-their-story/, Accessed: May 15, 2023, Sep. 2018.

[23] GitHub, *Differences between github apps and oauth apps*, https://docs.github.com/en/enterprise-server@3.5/apps/oauth-apps/building-oauth-apps/differences-between-github-apps-and-oauth-apps, Accessed: May 15, 2023.

[24] Mergifyio, *Mergify - github app*, https://github.com/marketplace/mergify, [Online; accessed 08-Feb-2023].

[25] AzureBoards, *Azure Boards - GitHub App*, https://github.com/marketplace/azure-boards, [Online; accessed 08-Feb-2023].

[26] D. Shekhar, *Workflow reporter - github app*, https://github.com/marketplace/workflow-reporter, [Online; accessed 08-Feb-2023].

[27] D. Messerschmitt, "Marketplace issues in software planning and design," *IEEE Software*, vol. 21, no. 3, pp. 62–70, 2004. DOI: 10.1109/MS.2004.1293074.

[28] AppDirect. "Digital marketplace." (), [Online]. Available: https://www.appdirect.com/resources/glossary/digital-marketplace (visited on 05/19/2023).

[29] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, and M. Harman, "App store effects on software engineering practices," *IEEE TSE*, 2021.

[30] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2016. DOI: 10.1109/MS.2015.153.

[31] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *Journal of Systems and Software*, vol. 125, pp. 207–219, 2017, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.11.027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216302291.

[32] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 108–111. DOI: 10.1109/MSR.2012.6224306.

[33] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, vol. 43, no. 9, pp. 817–847, 2017. DOI: 10.1109/TSE.2016.2630689.

[34] S. Jansen and E. Bloemendal, "Defining app stores: The role of curated marketplaces in software ecosystems," in *Software Business. From Physical Products to Software Services and Solutions*, G. Herzwurm and T. Margaria, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 195–206, ISBN: 978-3-642-39336-5.

[35] I. J. Mojica Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, "Examining the rating system used in mobile-app stores," *IEEE Software*, vol. 33, no. 6, pp. 86–92, 2016. DOI: 10.1109/MS.2015.56.

[36] M. Ali, M. E. Joorabchi, and A. Mesbah, "Same app, different app stores: A comparative study," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2017, pp. 79–90. DOI: 10.1109/MOBILESoft.2017.3.

[37] M. Nayebi, H. Farrahi, and G. Ruhe, "Analysis of marketed versus not-marketed mobile app releases," in *Proceedings of the 4th International Workshop on Release Engineering*, 2016, pp. 1–4.

[38] V. Wiemker, M. Neufeld, A. Bunova, *et al.*, "Digital assessment tools using animation features to quantify alcohol consumption: Systematic app store and literature review," *Journal of Medical Internet Research*, vol. 24, no. 3, e28927, 2022.

[39] E. Chen and E. R. Mangone, "A systematic review of apps using mobile criteria for adolescent pregnancy prevention (mcapp)," *JMIR Mhealth Uhealth*, vol. 4, no. 4, e122, Nov. 2016, ISSN: 2291-5222. DOI: 10.2196/mhealth.6611. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/27833070.

[40] A. Ciapponi, M. Donato, A. M. Gülmezoglu, T. Alconada, and A. Bardach, "Mobile apps for detecting falsified and substandard drugs: A systematic review," *PloS one*, vol. 16, no. 2, e0246061, 2021.

[41] N. Lau, A. O'Daffer, S. Colt, *et al.*, "Android and iphone mobile apps for psychosocial wellness and stress management: Systematic search in app stores and literature review," *JMIR mHealth and uHealth*, vol. 8, no. 5, e17798, 2020.

[42] S. Eis, O. Solà-Morales, A. Duarte-Dıaz, *et al.*, "Mobile applications in mood disorders and mental health: Systematic search in apple app store and google play store and review of the literature," *International Journal of Environmental Research and Public Health*, vol. 19, no. 4, p. 2186, 2022.

[43] D. Schultchen, Y. Terhorst, T. Holderied, *et al.*, "Stay present with your phone: A systematic review and standardized rating of mindfulness apps in european app stores," *International Journal of Behavioral Medicine*, vol. 28, pp. 552–560, 2021.

[44] L. J. Bry, T. Chou, E. Miguel, and J. S. Comer, "Consumer smartphone apps marketed for child and adolescent anxiety: A systematic review and content analysis," *Behavior therapy*, vol. 49, no. 2, pp. 249–261, 2018.

[45] T. Lieser, Y. Huang, and E. Sezgin, "The current state of mobile apps owned by large pediatric hospitals in the united states: Systematic search and analysis on google play and apple app stores," *JMIR Pediatr Parent*, vol. 5, no. 4, e38940, Oct. 2022, ISSN: 2561-6722. DOI: 10.2196/38940. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/36201385.

[46] I. H. Montano, I. de la Torre Dıez, R. López-Izquierdo, M. A. C. Villamor, and F. Martın-Rodrıguez, "Mobile triage applications: A systematic review in literature and play store," *Journal of Medical Systems*, vol. 45, no. 9, p. 86, 2021.

[47] K. Huckvale, S. Adomaviciute, J. T. Prieto, M. K.-S. Leow, and J. Car, "Smartphone apps for calculating insulin dose: A systematic assessment," *BMC medicine*, vol. 13, no. 1, pp. 1–10, 2015.

[48] J. Koepp, M. V. Baron, P. R. H. Martins, *et al.*, "The quality of mobile apps used for the identification of pressure ulcers in adults: Systematic survey and review of apps in app stores," *JMIR mHealth and uHealth*, vol. 8, no. 6, e14266, 2020.

[49] S. G. Saroar, W. Ahmed, and M. Nayebi, "Github marketplace for practitioners and researchers to date: A systematic analysis of the knowledge mobilization gap in open source software automation," *arXiv preprint arXiv:2208.00332*, 2022.

[50] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17, Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pp. 269–280, ISBN: 9781450345231. DOI: 10.1145/3029806.3029832. [Online]. Available: https://doi.org/10.1145/3029806.3029832.

[51] P. Liu, S. Ji, L. Fu, *et al.*, "Understanding the security risks of docker hub," in *Computer Security – ESORICS 2020*, Cham: Springer International Publishing, 2020, pp. 257–276, ISBN: 978-3-030-58951-6.

[52] C. Lin, S. Nadi, and H. Khazaei, "A large-scale data set and an empirical study of docker images hosted on docker hub," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 371–381. DOI: 10.1109/ICSME46990.2020.00043.

[53] N. Zhao, V. Tarasov, H. Albahar, *et al.*, "Large-scale analysis of the docker hub dataset," Sep. 2019, pp. 1–10. DOI: 10.1109/CLUSTER.2019.8891000.

[54] H. C. Jung and K.-K. Seo, "Prioritizing cloud service threats for succession to information security management system," *International Journal of Digital Contents and Applications for Smart Devices*, vol. 7, no. 1, pp. 1–8, 2020.

[55] W. Park and K. K. Seo, "A study on cloud-based software marketing strategies using cloud marketplace," *Journal of Logistics, Informatics and Service Science*, vol. 7, no. 2, pp. 1–13, 2020.

[56] A. Menychtas, A. Gatzioura, and T. Varvarigou, "A business resolution engine for cloud marketplaces," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 462–469. DOI: 10.1109/CloudCom.2011.68.

[57] B. Kitchenham, S. Pfleeger, and S. Lawrence, "Principles of survey research: Part 5: Populations and samples," *Software Eng. Notes*, 2002.

[58] C. Chandrasekara and P. Herath, "Getting started with github actions workflows," in *Hands-on GitHub Actions*, Springer, 2021.

[59] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?" In *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021.

[60] A. Decan, T. Mens, P. Mazrae, and M. Golzadeh, "On the use of github actions in software development repositories," in *IEEE ICSME*, 2022.

[61] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of ci services in github," in *IEEE SANER*, 2022.

[62] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[63] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, "Github actions: The impact on the pull request process," *arXiv preprint arXiv:2206.14118*, 2022.

[64] T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's supercharge the workflows: An empirical study of github actions," in *IEEE QRS-C, companion*, 2021.

[65] P. Valenzuela-Toledo and A. Bergel, "Evolution of github action workflows," in *IEEE SANER*, 2022.

[66] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *ACM ESEC/FSE*, 2015.

[67] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *IEEE/ACM ASE*, 2016.

[68] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: Assurance, security, and flexibility," in *ACM ESEC/FSE*, 2017.

[69] K. Gallaba and S. McIntosh, "Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis ci," *IEEE TSE*, 2020.

[70] C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, "Configuration smells in continuous delivery pipelines: A linter and a six-month study on gitlab," in *ACM ESEC/FSE*, 2020.

[71] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. Di Penta, "An empirical characterization of bad practices in continuous integration," *Empirical Software Engineering*, 2020.

[72] D. G. Widder, M. Hilton, C. Kästner, and B. Vasilescu, "I'm leaving you, travis: A continuous integration breakup story," in *MSR*, 2018.

[73] K. Gallaba, M. Lamothe, and S. McIntosh, "Lessons from eight years of operational data from a continuous integration service: An exploratory case study of circleci," in *ICSE*, 2022.

[74] F. Calefato, F. Lanubile, and L. Quaranta, "A preliminary investigation of MLOps practices in GitHub," in *IEEE ESEM*, 2022.

[75] G. Benedetti, L. Verderame, and A. Merlo, "Automatic security assessment of github actions workflows," in *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022.

[76] I. Koishybayev, A. Nahapetyan, R. Zachariah, *et al.*, "Characterizing the security of github CI workflows," in *USENIX Security 22*, 2022.

[77] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research: Part 3: Constructing a survey instrument," *Softw. Eng. Notes*, 2002.

[78] B. Kitchenham and S. Pfleeger, "Principles of survey research part 1: Turning lemons into lemonade," *Software Eng. Notes*, 2001.

[79] B. Kitchenham and S. L. Pfleeger, "Principles of survey research part 4: Questionnaire evaluation," *Softw. Eng. Notes*, 2002.

[80] R. Nisbett and T. Wilson, "The halo effect: Evidence for unconscious alteration of judgments," *Journal of Personality and Social Psychology*, 1977.

[81] E. Smith, R. Loftin, E. Murphy-Hill, and T. Zimmermann, "Improving developer participation rates in surveys," 2013.

[82] P. Edwards, I. Roberts, M. Clarke, *et al.*, "Increasing response rates to postal questionnaires: Systematic review," *BMJ (Clinical research ed.)*, 2002.

[83] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *ESEM*, IEEE, 2011.

[84] V. Braun and V. Clarke, *Thematic analysis.* American Psychological Association, 2012.

[85] B. Kitchenham and S. L. Pfleeger, "Principles of survey research part 6: Data analysis," *SIGSOFT Softw. Eng. Notes*, 2003.

[86] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," *ACM CSUR*, 2022.

[87] Y. Yang, X. Xia, D. Lo, T. Bi, J. Grundy, and X. Yang, "Predictive models in software engineering: Challenges and opportunities," *ACM TOSEM*, 2022.

[88] H. Borges and M. Tulio Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *JSS*, 2018.

[89] S. Wagner, D. Mendez, M. Felderer, D. Graziotin, and M. Kalinowski, "Challenges in survey research," in *Contemporary Empirical Methods in Software Engineering*, Springer International Publishing, 2020.

[90] M. Nayebi, B. Adams, and G. Ruhe, "Release practices for mobile apps–what do users and developers think?" In *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)*, IEEE, vol. 1, 2016, pp. 552–562.

[91] N. C. Shrikanth and T. Menzies, "Assessing practitioner beliefs about software defect prediction," in *Proceedings of the ACM/IEEE 42nd ICSE-SEIP*, 2020.

[92] C. Potts, "Software-engineering research revisited," *IEEE software*, vol. 10, no. 5, pp. 19–28, 1993.

[93] K. Beckman, N. Coulter, S. Khajenoori, and N. R. Mead, "Collaborations: Closing the industry-academia gap," *IEEE software*, vol. 14, no. 6, pp. 49–57, 1997.

[94] C. Wohlin, "Empirical software engineering research with industry: Top 10 challenges," in *2013 1st international workshop on conducting empirical studies in industry (CESI)*, IEEE, 2013, pp. 43–46.

[95] C. Wohlin and P. Runeson, "Guiding the selection of research methodology in industry–academia collaboration in software engineering," *Information and software technology*, vol. 140, p. 106 678, 2021.

[96] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "A model for technology transfer in practice," *IEEE software*, vol. 23, no. 6, pp. 88–95, 2006.

[97] F. Shull, M. G. Mendoncça, V. Basili, *et al.*, "Knowledge-sharing issues in experimental software engineering," *Empirical Software Engineering*, vol. 9, pp. 111–137, 2004.

[98] T. Mikkonen, C. Lassenius, T. Männistö, M. Oivo, and J. Järvinen, "Continuous and collaborative technology transfer: Software engineering research with real-time industry impact," *Information and Software Technology*, vol. 95, pp. 34–45, 2018.

[99] R. E. Santos, A. Bener, M. T. Baldassarre, C. V.C. Magalhães, J. S. Correia-Neto, and F. Q. da Silva, "Mind the gap: Are practitioners and researchers in software testing speaking the same language?" In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2019, pp. 10–17. DOI: 10.1109/CESSER-IP.2019.00010.

[100] K. Schmid, R. Rabiser, M. Becker, *et al.*, "Bridging the gap: Voices from industry and research on industrial relevance of splc," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*, ser. SPLC '21, Leicester, United Kingdom: Association for Computing Machinery, 2021, pp. 184–189, ISBN: 9781450384698. DOI: 10.1145/3461001.3474301. [Online]. Available: https://doi.org/10.1145/3461001.3474301.

[101] M. Ferreira, M. Bigonha, and K. A. M. Ferreira, *On the gap between software maintenance theory and practitioners' approaches*, 2021. arXiv: 2104.03824 [cs.SE].

[102] V. Garousi, M. Borg, and M. Oivo, "Practical relevance of software engineering research: Synthesizing the community's voice," *Empirical Software Engineering*, vol. 25, pp. 1687–1754, 2020.

[103] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, *How do software developers use github actions to automate their workflows?* 2021. DOI: 10.48550/ARXIV.2103.12224. [Online]. Available: https://arxiv.org/abs/2103.12224.

[104] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Softw. Engg.*, vol. 22, no. 6, pp. 3219–3253, Dec. 2017, ISSN: 1382-3256. DOI: 10.1007/s10664-017-9512-6. [Online]. Available: https://doi.org/10.1007/s10664-017-9512-6.

[105] A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, *On the Use of GitHub Actions in Software Development Repositories*, version 1.0.0, Jun. 2022. DOI: 10.5281/zenodo.6634682. [Online]. Available: https://doi.org/10.5281/zenodo.6634682.

[106] P. R. Mazrae, "Empirical analysis of the github actions ecosystem," 2022.

[107] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 767–778, ISBN: 9781450327565. DOI: 10.1145/2568225.2568263. [Online]. Available: https://doi.org/10.1145/2568225.2568263.

[108] J. Krüger, N. Corr, I. Schröter, and T. Leich, "Digging into the eclipse marketplace," Apr. 2017, pp. 60–65, ISBN: 978-3-319-57734-0. DOI: 10.1007/978-3-319-57735-7_7.

[109] J. Li, S. Beba, and M. M. Karlsen, "Evaluation of open-source ide plugins for detecting security vulnerabilities," *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019.

[110] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and software technology*, vol. 64, pp. 1–18, 2015.

[111] R. P. Buse and T. Zimmermann, "Information needs for software development analytics," in *2012 34th International Conference on Software Engineering (ICSE)*, IEEE, 2012, pp. 987–996.

[112] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 987–996. DOI: 10.1109/ICSE.2012.6227122.

[113] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements engineering*, vol. 11, no. 1, pp. 102–107, 2006.

[114] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian, "Exploiting enriched contextual information for mobile app classification," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 1617–1621.

[115] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[116] B. Vasilescu, A. Serebrenik, and M. Brand, "The babel of software development: Linguistic diversity in open source," vol. 8238, Nov. 2013, pp. 391–404, ISBN: 978-3-319-03259-7. DOI: 10.1007/978-3-319-03260-3_34.

[117] A. A. Al-Subaihin, F. Sarro, S. Black, *et al.*, "Clustering mobile apps based on mined textual features," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16, Ciudad Real, Spain: Association for Computing Machinery, 2016, ISBN: 9781450344272. DOI: 10.1145/2961111.2962600. [Online]. Available: https://doi.org/10.1145/2961111.2962600.

[118] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 12–23.

[119] D. M. Fernández, S. Wagner, M. Kalinowski, *et al.*, "Naming the pain in requirements engineering," *Empirical software engineering*, vol. 22, no. 5, pp. 2298–2338, 2017.

[120] S. G. Saroar and M. Nayebi, "Developers' perception of github actions: A survey analysis," ser. EASE '23, Oulu, Finland: Association for Computing Machinery, 2023, pp. 121–130, ISBN: 9798400700446. DOI: 10.1145/3593434.3593475. [Online]. Available: https://doi.org/10.1145/3593434.3593475.

[121] A. D. Sorbo, J. Spillner, G. Canfora, and S. Panichella, ""won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on github," *CoRR*, vol. abs/1904.02414, 2019. arXiv: 1904.02414. [Online]. Available: http://arxiv.org/abs/1904.02414.

[122] V. Lenarduzzi, D. Taibi, D. Tosi, L. Lavazza, and S. Morasca, "Open source software evaluation, selection, and adoption: A systematic literature review," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2020, pp. 437–444.

[123] O. Franco-Bedoya, D. Ameller, D. Costal, and X. Franch, "Open source software ecosystems: A systematic mapping," *Information and software technology*, vol. 91, pp. 160–185, 2017.

[124] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A systematic mapping study of software development with github," *IEEE Access*, vol. 5, pp. 7173–7192, 2017.

[125] 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.

[126] X. Franch, D. M. Fernández, M. Oriol, *et al.*, "How do practitioners perceive the relevance of requirements engineering research? an ongoing study," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, IEEE, 2017, pp. 382–387.

[127] P. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, IEEE, 2016, pp. 108–119.

[128] P. Devanbu, T. Zimmermann, and C. Bird, "Belief and evidence: How software engineers form their opinions," *IEEE Software*, vol. 35, no. 06, pp. 72–76, 2018.

[129] N. Shrikanth, W. Nichols, F. M. Fahid, and T. Menzies, "Assessing practitioner beliefs about software engineering," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–32, 2021.

[130] M. Nayebi and G. Ruhe, "Optimized functionality for super mobile apps," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 388–393. DOI: 10.1109/RE.2017.72.

Appendix A: Categorized Papers

Table A.1: Papers mapped to GitHub Marketplace categories

Category	Paper ID
API management	P13, P22, P52, P126, P172, P245, P322, P340, P372, P383, P454, P455, P498
Testing	P11, P15, P21, P49, P59, P64, P67, P71, P76, P78, P79, P82, P94, P101, P114, P116, P117, P118, P123, P124, P146, P150, P154, P156, P159, P162, P163, P165, P171, P174, P178, P179, P189, P195, P203, P207, P208, P212, P234, P244, P246, P247, P248, P250, P257, P260, P261, P263, P266, P267, P268, P272, P286, P305, P313, P323, P330, P337, P339, P342, P343, P348, P352, P353, P354, P357, P361, P362, P363, P364, P380, P384, P398, P417, P425, P426, P428, P432, P437, P438, P441, P442, P444, P450, P458, P459, P462, P464, P466, P470, P473, P474, P476, P480, P483, P487, P489, P492, P493, P496, P497, P502, P510, P515
Utilities	P8, P10, P28, P30, P34, P41, P51, P52, P61, P87, P90, P95, P103, P106, P116, P125, P126, P136, P137, P140, P141, P145, P149, P150, P152, P154, P158, P161, P175, P180, P181, P193, P216, P221, P243, P251, P254, P284, P322, P349, P394, P397, P403, P405, P418, P448, P463, P468, P494
Reporting	P16, P35, P36, P40, P53, P55, P57, P59, P69, P86, P87, P92, P96, P160, P166, P168, P202, P206, P219, P224, P225, P271, P309, P316, P330, P343, P350, P351, P352, P356, P413, P418, P468, P477, P484, P493, P503
Continuous integration	P29, P71, P85, P180, P210, P348, P350, P353, P368, P369, P374, P376, P385, P386, P388, P480
Publishing	P6, P19, P54, P85, P107, P172, P180, P201, P223, P227, P273, P275, P276, P287, P288, P296, P335, P336

Continued on next page

Table A.1 – continued from previous page

Category	paper ID
Support	P8, P12, P18, P21, P34, P38, P45, P60, P65, P68, P70, P72, P74, P85, P88, P92, P94, P103, P105, P107, P112, P121, P143, P154, P155, P156, P161, P183, P193, P201, P202, P214, P225, P227, P233, P249, P251, P256, P259, P270, P274, P278, P284, P288, P294, P310, P326, P359, P388, P406, P408, P424, P428, P431, P434, P439, P441, P454, P460, P461, P468, P471, P472, P475, P479, P488, P490, P491, P504, P506, P512
Project management	P5, P6, P9, P12, P16, P19, P29, P37, P47, P57, P75, P102, P107, P113, P121, P134, P138, P144, P168, P173, P223, P229, P233, P236, P240, P241, P243, P249, P253, P255, P262, P270, P273, P274, P275, P276, P281, P283, P296, P304, P312, P316, P325, P326, P332, P335, P336, P354, P362, P366, P373, P374, P404, P405, P418, P457, P460
Code review	P1, P2, P17, P22, P28, P41, P48, P56, P58, P66, P68, P70, P80, P86, P91, P93, P98, P103, P108, P111, P119, P126, P130, P141, P142, P151, P152, P173, P191, P215, P218, P232, P235, P273, P282, P297, P300, P345, P347, P349, P352, P356, P357, P358, P359, P360, P364, P365, P367, P368, P372, P374, P375, P377, P378, P379, P381, P387, P396, P416, P420, P421, P436, P451, P452, P453, P454, P471, P487, P491
Deployment	P9, P67, P149, P175, P275, P276, P294, P308
Chat	P27, P31, P62, P206, P221, P382, P389
Community	P5, P19, P20, P27, P29, P31, P37, P40, P48, P60, P61, P62, P81, P100, P104, P105, P120, P227, P259, P288, P346, P360, P366, P370, P377, P381, P408, P427
Dependency management	P7, P66, P129, P145, P147, P157, P166, P179, P182, P187, P188, P201, P207, P211, P213, P215, P220, P228, P237, P238, P239, P388, P409, P413, P419, P463, P484, P505
AI Assisted	P9, P14, P17, P28, P46, P50, P53, P63, P69, P79, P89, P99, P113, P122, P128, P132, P133, P139, P141, P143, P144, P147, P149, P152, P157, P168, P181, P198, P211, P212, P213, P217, P220, P229, P233, P234, P235, P236, P237, P238, P239, P240, P246, P250, P274, P281, P283, P290, P294, P299, P307, P318, P328, P334, P345, P346, P355, P356, P358, P363, P367, P383, P390, P394, P397, P399, P400, P401, P402, P403, P405, P415, P423, P429, P435, P456, P461, P474, P485, P495, P498, P503, P508, P513

Continued on next page

Table A.1 – continued from previous page

Category	paper ID
Open Source management	P11, P22, P37, P47, P60, P61, P62, P65, P71, P72, P81, P95, P104, P105, P134, P206, P209, P221, P229, P360, P367, P371, P387, P391, P404, P427, P504
Security	P45, P122, P134, P158, P177, P182, P187, P263, P282, P298, P340, P347, P351, P354, P357, P359, P369, P373, P380, P410, P435, P440, P443, P446, P449, P469, P476, P477, P486, P496, P499, P501
Monitoring	P68, P97, P102, P110, P118, P125, P129, P137, P138, P144, P153, P157, P160, P163, P182, P236, P249, P254, P262, P308, P378, P392, P393, P395, P424, P465, P494
Code quality	P1, P2, P13, P21, P35, P49, P51, P56, P58, P63, P66, P76, P77, P80, P82, P84, P88, P89, P90, P91, P93, P96, P98, P99, P101, P106, P108, P110, P111, P113, P115, P119, P123, P125, P128, P129, P130, P131, P132, P133, P136, P137, P139, P140, P142, P143, P146, P151, P153, P156, P158, P164, P166, P167, P170, P173, P174, P175, P179, P181, P189, P190, P192, P200, P203, P209, P210, P214, P215, P216, P223, P232, P238, P242, P250, P252, P256, P257, P262, P263, P265, P271, P272, P277, P285, P289, P292, P295, P296, P297, P300, P305, P312, P314, P327, P331, P344, P345, P347, P351, P353, P355, P361, P364, P365, P372, P375, P376, P379, P383, P385, P387, P411, P414, P415, P416, P422, P430, P433, P434, P436, P437, P438, P441, P445, P447, P467, P470, P471, P477, P478, P481, P482, P485, P506, P507, P508, P511, P514, P515
Localization	P13, P131, P132, P133, P136, P147, P188, P207, P211, P213, P220, P237, P239, P371, P412
Desktop tools	P4, P87, P95, P155, P225
Mobile	P40, P64, P67, P96, P114, P169, P244, P439, P443
IDEs	P34, P41, P191, P349, P361, P431, P448, P465, P472, P509
Mobile CI	P64
Code search	P1, P7, P11, P17, P20, P35, P52, P56, P58, P77, P80, P82, P88, P91, P93, P99, P101, P106, P111, P119, P123, P139, P140, P142, P150, P151, P171, P183, P195, P198, P224, P248, P252, P256, P257, P285, P355, P358, P363, P407, P461, P464, P472, P487, P491, P500
Code scanning ready	P2, P77, P89, P128, P131, P188, P195, P202, P224, P251, P272, P282, P365, P369, P376, P378, P379, P380, P463, P464, P508

Continued on next page

Table A.1 – continued from previous page

Category	paper ID
Learning	P5, P36, P108, P112, P160, P191, P259, P278, P309, P343, P366, P368, P370
Time tracking	P14, P47, P138, P235

The mapped studies into each category of GitHub marketplace. For answering **RQ2**, two authors manually mapped the studies into the schemas introduced. For a side by side comparison in **RQ3**, we used the GitHub categories. Table A.1 shows the results of this comparison.

Appendix B: List of the Mapped Studies

[P1] Robles, G., Gonzalez-Barahona, J. M., & Merelo, J. J. (2006). Beyond source code: the importance of other artifacts in software development (a case study). Journal of Systems and Software, 79(9), 1233-1248.

[P2] Souter, A. L., & Pollock, L. L. (2002). Characterization and automatic identification of type infeasible call chains. Information and Software Technology, 44(13), 721-732.

[P3] Tolosa, J. B., Sanjuán-Martínez, O., García-Díaz, V., G-Bustelo, B. C. P., & Lovelle, J. M. C. (2011). Towards the systematic measurement of ATL transformation models. Software: Practice and Experience, 41(7), 789-815.

[P4] Dalton, A. R., & Hallstrom, J. O. (2009). nAIT: A source analysis and instrumentation framework for nesC. Journal of Systems and Software, 82(7), 1057-1072.

[P5] Uhl, A. (2008). Model-driven development in the enterprise. IEEE software, 25(1), 46-49.

[P6] Lago, P., Muccini, H., & Van Vliet, H. (2009). A scoped approach to traceability management. Journal of Systems and Software, 82(1), 168-182.

[P7] Strasunskas, D., & Hakkarainen, S. E. (2012). Domain model-driven software engineering: A method for discovery of dependency links. Information and Software Technology, 54(11), 1239-1249.

[P8] Lizcano, D., Alonso, F., Soriano, J., & López, G. (2015). Automated end user-centred adaptation of web components through automated description logic-based reasoning. Information and Software Technology, 57, 446-462.

[P9] Patel, P., & Cassou, D. (2015). Enabling high-level application development for the Internet of Things. Journal of Systems and Software, 103, 62-84.

[P10] Gallardo, M. M., Merino, P., Panizo, L., & Linares, A. (2011). A practical use of model checking for synthesis: generating a dam controller for flood management. Software: Practice and Experience, 41(11), 1329-1347.

[P11] Smith, B. H., & Williams, L. (2009). On guiding the augmentation of an automated test suite via mutation analysis. Empirical software engineering, 14(3), 341-369.

[P12] Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., & Tamburrelli, G. (2010). Dynamic QoS management and optimization in service-based systems. IEEE Transactions on Software Engineering, 37(3), 387-409.

[P13] Wang, X., Zhang, L., Xie, T., Mei, H., & Sun, J. (2012). Locating need-to-externalize constant strings for software internationalization with generalized string-taint analysis. IEEE Transactions on Software Engineering, 39(4), 516-536.

[P14] De Caso, G., Braberman, V., Garbervetsky, D., & Uchitel, S. (2010). Automated abstractions for contract validation. IEEE Transactions on Software Engineering, 38(1), 141-162.

[P15] Kasoju, A., Petersen, K., & Mäntylä, M. V. (2013). Analyzing an automotive testing process with evidence-based software engineering. Information and Software Technology, 55(7), 1237-1259.

[P16] Winter, J., Rönkkö, K., & Rissanen, M. (2014). Identifying organizational barriers—a case study of usability work when developing software in the automation industry. Journal of Systems and Software, 88, 54-73.

[P17] White, J., Doughtery, B., & Schmidt, D. C. (2010). Ascent: An algorithmic technique for designing hardware and software in tandem. IEEE Transactions on Software Engineering, 36(6), 838-851.

[P18] Quirchmayr, T., Paech, B., Kohl, R., Karey, H., & Kasdepke, G. (2018). Semi-automatic rule-based domain terminology and software feature-relevant information extraction from natural language user manuals. Empirical Software Engineering, 23(6), 3630-3683.

[P19] Zhang, J., Wang, Y., & Xie, T. (2019). Software feature refinement prioritization based on online user review mining. Information and Software Technology, 108, 30-34.

[P20] Colanzi, T. E., Vergilio, S. R., Assuncao, W. K. G., & Pozo, A. (2013). Search based software engineering: Review and analysis of the field in Brazil. Journal of Systems and Software, 86(4), 970-984.

[P21] Grano, G., Palomba, F., Di Nucci, D., De Lucia, A., & Gall, H. C. (2019). Scented since the beginning: On the diffuseness of test smells in automatically generated test code. Journal of Systems and Software, 156, 312-327.

[P22] Kim, M., Notkin, D., Grossman, D., & Wilson, G. (2012). Identifying and summarizing systematic code changes via rule inference. IEEE Transactions on Software Engineering, 39(1), 45-62.

[P23] Faragardi, H. R., Lisper, B., Sandström, K., & Nolte, T. (2018). A resource efficient framework to run automotive embedded software on multi-core ECUs. Journal of Systems and Software, 139, 64-83.

[P24] Švogor, I., Crnković, I., & Vrček, N. (2019). An extensible framework for software configuration optimization on heterogeneous computing systems: Time and energy case study. Information and software technology, 105, 30-42.

[P25] Gacitúa, R., Sepúlveda, S., & Mazo, R. (2019). FM-CF: A framework for classifying feature model building approaches. Journal of Systems and Software, 154, 1-21.

[P26] Agh, H., & Ramsin, R. (2016). A pattern-based model-driven approach for situational method engineering. Information and Software Technology, 78, 95-120.

[P27] Mohamad, M., Liebel, G., & Knauss, E. (2017). LoCo CoCo: Automatically constructing coordination and communication networks from model-based systems engineering data. Information and Software Technology, 92, 179-193.

[P28] Falessi, D., Di Penta, M., Canfora, G., & Cantone, G. (2017). Estimating the number of remaining links in traceability recovery. Empirical Software Engineering, 22(3), 996-1027.

[P29] Ståhl, D., Mårtensson, T., & Bosch, J. (2017). The continuity of continuous integration: Correlations and consequences. Journal of Systems and Software, 127, 150-167.

[P30] Borg, M., Wnuk, K., Regnell, B., & Runeson, P. (2016). Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. IEEE Transactions on Software Engineering, 43(7), 675-700.

[P31] Barua, A., Thomas, S. W., & Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering, 19(3), 619-654.

[P32] Jagroep, E., van der Ent, A., van der Werf, J. M. E., Hage, J., Blom, L., van Vliet, R., & Brinkkemper, S. (2018). The hunt for the guzzler: Architecture-based energy profiling using stubs. Information and Software Technology, 95, 165-176.

[P33] Ali, N. B., & Usman, M. (2018). Reliability of search in systematic reviews: Towards a quality assessment framework for the automated-search strategy. Information and Software Technology, 99, 133-147.

[P34] Osvaldo Jr, S. S., Lopes, D., Silva, A. C., & Abdelouahab, Z. (2017). Developing software systems to Big Data platform based on MapReduce model: An approach based on Model Driven Engineering. Information and Software Technology, 92, 30-48.

[P35] Li, H., Shang, W., Zou, Y., & E Hassan, A. (2017). Towards just-in-time suggestions for log changes. Empirical Software Engineering, 22(4), 1831-1865.

[P36] Ebert, C., Heidrich, J., Martínez-Fernández, S., & Trendowicz, A. (2019). Data science: technologies for better software. IEEE software, 36(6), 66-72.

[P37] Jiang, J., Lo, D., Ma, X., Feng, F., & Zhang, L. (2017). Understanding inactive yet available assignees in GitHub. Information and Software Technology, 91, 44-55.

[P38] Xiong, Y., Zhang, H., Hubaux, A., She, S., Wang, J., & Czarnecki, K. (2014). Range fixes: Interactive error resolution for software configuration. Ieee transactions on software engineering, 41(6), 603-619.

[P39] Hamilton, M. H. (2018). What the Errors Tell Us. IEEE Software, 35(5), 32-37.

[P40] Martens, D., & Maalej, W. (2019). Towards understanding and detecting fake reviews in app stores. Empirical Software Engineering, 24(6), 3316-3355.

[P41] Hübner, P., & Paech, B. (2020). Interaction-based creation and maintenance of continuously usable trace links between requirements and source code. Empirical Software Engineering, 25(5), 4350-4377.

[P42] Carbonnel, J., Huchard, M., & Nebut, C. (2019). Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. Journal of Systems and Software, 152, 1-23.

[P43] Barricelli, B. R., Cassano, F., Fogli, D., & Piccinno, A. (2019). End-user development, end-user programming and end-user software engineering: A systematic mapping study. Journal of Systems and Software, 149, 101-137.

[P44] Ciolek, D., Braberman, V., D'Ippolito, N., Piterman, N., & Uchitel, S. (2016). Interaction models and automated control under partial observable environments. IEEE Transactions on Software Engineering, 43(1), 19-33.

[P45] Caivano, D., Fogli, D., Lanzilotti, R., Piccinno, A., & Cassano, F. (2018). Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. Journal of Systems and Software, 144, 295-313.

[P46] Tawosi, V., Jalili, S., & Hasheminejad, S. M. H. (2015). Automated software design using ant colony optimization with semantic network support. Journal of Systems and Software, 109, 1-17.

[P47] Rakha, M. S., Shang, W., & Hassan, A. E. (2016). Studying the needed effort for identifying duplicate issues. Empirical Software Engineering, 21(5), 1960-1989.

[P48] Sharma, S., & Sodhi, B. (2019). Using Stack Overflow content to assist in code review. Software: Practice and Experience, 49(8), 1255-1277.

[P49] Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., ... & Wortmann, A. (2019). SMArDT modeling for automotive software testing. Software: Practice and Experience, 49(2), 301-328.

[P50] Islam, M. R., & Zibran, M. F. (2018). SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. Journal of Systems and Software, 145, 125-146.

[P51] Cornu, B., Barr, E. T., Seinturier, L., & Monperrus, M. (2016). Casper: Automatic tracking of null dereferences to inception with causality traces. Journal of Systems and Software, 122, 52-62.

[P52] Zhang, F., Niu, H., Keivanloo, I., & Zou, Y. (2017). Expanding queries for code search using semantically related api class-names. IEEE Transactions on Software Engineering, 44(11), 1070-1082.

[P53] Bao, L., Li, J., Xing, Z., Wang, X., Xia, X., & Zhou, B. (2017). Extracting and analyzing time-series HCI data from screen-captured task videos. Empirical Software Engineering, 22(1), 134-174.

[P54] Nayebi, M., Ruhe, G., & Zimmermann, T. (2019). Mining treatment-outcome constructs from sequential software engineering data. IEEE Transactions on Software Engineering, 47(2), 393-411.

[P55] Haghighatkhah, A., Oivo, M., Banijamali, A., & Kuvaja, P. (2017). Improving the state of automotive software engineering. IEEE Software, 34(5), 82-86.

[P56] Li, H., Shang, W., & Hassan, A. E. (2017). Which log level should developers choose for a new logging statement?. Empirical Software Engineering, 22(4), 1684-1716.

[P57] Zou, W., Lo, D., Chen, Z., Xia, X., Feng, Y., & Xu, B. (2018). How practitioners perceive automated bug report management techniques. IEEE Transactions on Software Engineering, 46(8), 836-862.

[P58] Mahmoud, A., & Bradshaw, G. (2017). Semantic topic models for source code analysis. Empirical Software Engineering, 22(4), 1965-2000.

[P59] Arcuri, A. (2018). An experience report on applying software testing academic results in industry: we need usable automated test generation. Empirical Software Engineering, 23(4), 1959-1981.

[P60] Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., & Zhang, L. (2017). Why and how developers fork what from whom in GitHub. Empirical Software Engineering, 22(1), 547-578.

[P61] Gousios, G., & Spinellis, D. (2014). Conducting quantitative software engineering studies with Alitheia Core. Empirical Software Engineering, 19(4), 885-925.

[P62] Huang, Q., Xia, X., Lo, D., & Murphy, G. C. (2018). Automating intention mining. IEEE Transactions on Software Engineering, 46(10), 1098-1119.

[P63] AlOmar, E. A., Mkaouer, M. W., & Ouni, A. (2021). Toward the automatic classification of self-affirmed refactoring. Journal of Systems and Software, 171, 110821.

[P64] Silva, D. B., Eler, M. M., Durelli, V. H., & Endo, A. T. (2018). Characterizing mobile apps from a source and test code viewpoint. Information and Software Technology, 101, 32-50.

[P65] LaToza, T. D., Di Lecce, A., Ricci, F., Towne, W. B., & Van der Hoek, A. (2018). Microtask programming. IEEE Transactions on Software Engineering, 45(11), 1106-1124.

[P66] de Oliveira, M. C., Freitas, D., Bonifácio, R., Pinto, G., & Lo, D. (2019). Finding needles in a haystack: Leveraging co-change dependencies to recommend refactorings. Journal of Systems and Software, 158, 110420.

[P67] Cruz, L., Abreu, R., & Lo, D. (2019). To the attention of mobile software developers: guess what, test your app!. Empirical Software Engineering, 24(4), 2438-2468.

[P68] Nelson, N., Brindescu, C., McKee, S., Sarma, A., & Dig, D. (2019). The life-cycle of merge conflicts: processes, barriers, and strategies. Empirical Software Engineering, 24(5), 2863-2906.

[P69] Zhou, P., Liu, J., Liu, X., Yang, Z., & Grundy, J. (2019). Is deep learning better than traditional approaches in tag recommendation for software information sites?. Information and software technology, 109, 1-13.

[P70] Zanjani, M. B., Kagdi, H., & Bird, C. (2015). Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering, 42(6), 530-543.

[P71] Munir, H., Linåker, J., Wnuk, K., Runeson, P., & Regnell, B. (2018). Open innovation using open source tools: A case study at Sony Mobile. Empirical Software Engineering, 23(1), 186-223.

[P72] Sbai, N., Lenarduzzi, V., Taibi, D., Sassi, S. B., & Ghezala, H. H. B. (2018). Exploring information from OSS repositories and platforms to support OSS selection decisions. Information and Software Technology, 104, 104-108.

[P73] Graaf, B., Lormans, M., & Toetenel, H. (2003). Embedded software engineering: the state of the practice. IEEE software, 20(6), 61-69.

[P74] Krall, J., Menzies, T., & Davies, M. (2015). Gale: Geometric active learning for search-based software engineering. IEEE Transactions on Software Engineering, 41(10), 1001-1018.

[P75] Di Sorbo, A., Canfora, G., & Panichella, S. (2019). ' Won't We Fix this Issue?' Qualitative Characterization and Automated Identification of Wontfix Issues on GitHub. arXiv preprint arXiv:1904.02414.

[P76] Zhang, M., Li, Y., Li, X., Chen, L., Zhang, Y., Zhang, L., & Khurshid, S. (2019). An empirical study of boosting spectrum-based fault localization via pagerank. IEEE Transactions on Software Engineering, 47(6), 1089-1113.

[P77] Soremekun, E., Kirschner, L., Böhme, M., & Zeller, A. (2021). Locating faults with program slicing: an empirical analysis. Empirical Software Engineering, 26(3), 1-45.

[P78] Kazerouni, A. M., Davis, J. C., Basak, A., Shaffer, C. A., Servant, F., & Edwards, S. H. (2021). Fast and accurate incremental feedback for students' software tests using selective mutation analysis. Journal of Systems and Software, 175, 110905.

[P79] Ghosh, D., & Singh, J. (2021). Spectrum-based multi-fault localization using Chaotic Genetic Algorithm. Information and Software Technology, 133, 106512.

[P80] Almhana, R., Kessentini, M., & Mkaouer, W. (2021). Method-level bug localization using hybrid multi-objective search. Information and Software Technology, 131, 106474.

[P81] Tamburri, D. A., Palomba, F., & Kazman, R. (2019). Exploring community smells in open-source: An automated approach. IEEE Transactions on software Engineering, 47(3), 630-652.

[P82] Xiaobo, Y., Bin, L., & Shihai, W. (2021). A Test Restoration Method based on Genetic Algorithm for effective fault localization in multiple-fault programs. Journal of Systems and Software, 172, 110861.

[P83] Panichella, A. (2021). A Systematic Comparison of search-Based approaches for LDA hyperparameter tuning. Information and Software Technology, 130, 106411.

[P84] Ye, H., Martinez, M., Durieux, T., & Monperrus, M. (2021). A comprehensive study of automatic program repair on the QuixBugs benchmark. Journal of Systems and Software, 171, 110825.

[P85] Saidani, I., Ouni, A., Chouchen, M., & Mkaouer, M. W. (2020). Predicting continuous integration build failures using evolutionary search. Information and Software Technology, 128, 106392.

[P86] Panichella, S., & Zaugg, N. (2020). An empirical investigation of relevant changes and automation needs in modern code review. Empirical Software Engineering, 25(6), 4833-4872.

[P87] Wei, R., Zolotas, A., Hoyos Rodriguez, H., Gerasimou, S., Kolovos, D. S., & Paige, R. F. (2020). Automatic generation of UML profile graphical editors for Papyrus. Software and Systems Modeling, 19(5), 1083-1106.

[P88] Alizadeh, V., Kessentini, M., Mkaouer, M. W., Ocinneide, M., Ouni, A., & Cai, Y. (2018). An interactive and dynamic search-based approach to software refactoring recommendations. IEEE Transactions on Software Engineering, 46(9), 932-961.

[P89] Zhang, Y., Jin, D., Xing, Y., & Gong, Y. (2020). Automated defect identification via path analysis-based features with transfer learning. Journal of Systems and Software, 166, 110585.

[P90] Koyuncu, A., Liu, K., Bissyandé, T. F., Kim, D., Klein, J., Monperrus, M., & Le Traon, Y. (2020). Fixminer: Mining relevant fix patterns for automated program repair. Empirical Software Engineering, 25(3), 1980-2024.

[P91] de Freitas Farias, M. A., de Mendonça Neto, M. G., Kalinowski, M., & Spínola, R. O. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. Information and Software Technology, 121, 106270.

[P92] Chen, D., & Zhang, R. (2020). Catla-HS: An Open Source Project for Tuning and Analyzing MapReduce Performance on Hadoop and Spark. IEEE Software.

[P93] Janke, M., & Mader, P. (2020). Graph Based Mining of Code Change Patterns from Version Control Commits. IEEE Transactions on Software Engineering.

[P94] Guizzo, G., Sarro, F., Krinke, J., & Vergilio, S. R. (2020). Sentinel: A hyper-heuristic for the generation of mutant reduction strategies. IEEE Transactions on Software Engineering.

[P95] Zolotas, A., Hoyos Rodriguez, H., Hutchesson, S., Sanchez Pina, B., Grigg, A., Li, M., ... & Paige, R. F. (2020). Bridging proprietary modelling and open-source model management tools: the case of PTC integrity modeller and epsilon. Software and Systems Modeling, 19(1), 17-38.

[P96] Zeng, Y., Chen, J., Shang, W., & Chen, T. H. P. (2019). Studying the characteristics of logging practices in mobile apps: a case study on f-droid. Empirical Software Engineering, 24(6), 3394-3434.

[P97] Avelino, G., Passos, L., Petrillo, F., & Valente, M. T. (2018). Who Can Maintain This Code?: Assessing the Effectiveness of Repository-Mining Techniques for Identifying Software Maintainers. IEEE Software, 36(6), 34-42.

[P98] Mohan, M., & Greer, D. (2019). Using a many-objective approach to investigate automated refactoring. Information and Software Technology, 112, 83-101.

[P99] Chen, H., Huang, Y., Liu, Z., Chen, X., Zhou, F., & Luo, X. (2019). Automatically detecting the scopes of source code comments. Journal of Systems and Software, 153, 45-63.

[P100] Tamburri, D. A., Palomba, F., Serebrenik, A., & Zaidman, A. (2019). Discovering community patterns in open-source: a systematic approach and its evaluation. Empirical Software Engineering, 24(3), 1369-1417.

[P101] Xie, X., Chen, B., Zou, L., Liu, Y., Le, W., & Li, X. (2017). Automatic loop summarization via path dependency analysis. IEEE Transactions on Software Engineering, 45(6), 537-557.

[P102] Choetkiertikul, M., Dam, H. K., Tran, T., Ghose, A., & Grundy, J. (2017). Predicting delivery capability in iterative software development. IEEE Transactions on Software Engineering, 44(6), 551-573.

[P103] Li, Y., Zhu, C., Rubin, J., & Chechik, M. (2017). Semantic slicing of software version histories. IEEE Transactions on Software Engineering, 44(2), 182-201.

[P104] Koo, H. M., & Ko, I. Y. (2017). Construction and utilization of problem-solving knowledge in open source software environments. Journal of Systems and Software, 131, 402-418.

[P105] Kapitsaki, G. M., Kramer, F., & Tselikas, N. D. (2017). Automating the license compatibility process in open source software with SPDX. Journal of systems and software, 131, 386-401.

[P106] Morales, R., Soh, Z., Khomh, F., Antoniol, G., & Chicano, F. (2017). On the use of developers' context for automatic refactoring of software anti-patterns. Journal of systems and software, 128, 236-251.

[P107] Choetkiertikul, M., Dam, H. K., Tran, T., & Ghose, A. (2017). Predicting the delay of issues with due dates in software projects. Empirical Software Engineering, 22(3), 1223-1263.

[P108] Beller, M., Zaidman, A., Karpov, A., & Zwaan, R. A. (2017). The last line effect explained. Empirical Software Engineering, 22(3), 1508-1536.

[P109] Xiao, L., Yu, Z., Chen, B., & Wang, X. (2017). How robust is your development team?. IEEE Software, 35(1), 64-71.

[P110] Tian, Y., Ali, N., Lo, D., & Hassan, A. E. (2016). On the unreliability of bug severity data. Empirical Software Engineering, 21(6), 2298-2323.

[P111] Mohan, M., Greer, D., & McMullan, P. (2016). Technical debt reduction using search based automated refactoring. Journal of Systems and Software, 120, 183-194.

[P112] Fu, W., Menzies, T., & Shen, X. (2016). Tuning for software analytics: Is it really necessary?. Information and Software Technology, 76, 135-146.

[P113] Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., & Runeson, P. (2016). Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empirical Software Engineering, 21(4), 1533-1578.

[P114] Amalfitano, D., Fasolino, A. R., Tramontana, P., Ta, B. D., & Memon, A. M. (2014). MobiGUITAR: Automated model-based testing of mobile apps. IEEE software, 32(5), 53-59.

[P115] Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., & Hamdi, M. S. (2015). Improving multi-objective code-smells correction using development history. Journal of Systems and Software, 105, 18-39.

[P116] Fraser, G., & Arcuri, A. (2015). 1600 faults in 100 projects: automatically finding faults while achieving high coverage with evosuite. Empirical software engineering, 20(3), 611-639.

[P117] Iqbal, M. Z., Arcuri, A., & Briand, L. (2015). Environment modeling and simulation for automated testing of soft real-time embedded software. Software & Systems Modeling, 14(1), 483-524.

[P118] Cornu, B., Seinturier, L., & Monperrus, M. (2015). Exception handling analysis and transformation using fault injection: Study of resilience against unanticipated exceptions. Information and Software Technology, 57, 66-76.

[P119] Insa, D., & Silva, J. (2015). Automatic transformation of iterative loops into recursive methods. Information and Software Technology, 58, 95-109.

[P120] Schall, D. (2014). Who to follow recommendation in large-scale online development communities. Information and Software Technology, 56(12), 1543-1555.

[P121] Alam, O., Adams, B., & Hassan, A. E. (2012). Preserving knowledge in software projects. Journal of systems and software, 85(10), 2318-2330.

[P122] Shahmehri, N., Mammar, A., De Oca, E. M., Byers, D., Cavalli, A., Ardi, S., & Jimenez, W. (2012). An advanced approach for modeling and detecting software vulnerabilities. Information and Software Technology, 54(9), 997-1013.

[P123] Woungang, I., Akinladejo, F. O., White, D. W., & Obaidat, M. S. (2012). Coding-error based defects in enterprise resource planning software: Prevention, discovery, elimination and mitigation. Journal of systems and software, 85(7), 1682-1698.

[P124] Zhang, L., Ma, X., Lu, J., Xie, T., Tillmann, N., & De Halleux, P. (2011). Environmental modeling for automated cloud application testing. IEEE software, 29(2), 30-35.

[P125] Deissenboeck, F., Juergens, E., Hummel, B., Wagner, S., y Parareda, B. M., & Pizka, M. (2008). Tool support for continuous quality control. IEEE software, 25(5), 60-67.

[P126] Xing, Z., & Stroulia, E. (2007). API-evolution support with Diff-CatchUp. IEEE Transactions on Software Engineering, 33(12), 818-836.

[P127] Poshyvanyk, D., Gueheneuc, Y. G., Marcus, A., Antoniol, G., & Rajlich, V. (2007). Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Transactions on Software Engineering, 33(6), 420-432.

[P128] Demsky, B., & Rinard, M. C. (2006). Goal-directed reasoning for specification-based data structure repair. IEEE Transactions on Software Engineering, 32(12), 931-951.

[P129] Di Penta, M., Neteler, M., Antoniol, G., & Merlo, E. (2005). A language-independent software renovation framework. Journal of Systems and Software, 77(3), 225-240.

[P130] Tsantalis, N., Chatzigeorgiou, A., & Stephanides, G. (2005). Predicting the probability of change in object-oriented systems. IEEE Transactions on Software Engineering, 31(7), 601-614.

[P131] Williams, C. C., & Hollingsworth, J. K. (2005). Automatic mining of source code repositories to improve bug finding techniques. IEEE Transactions on Software Engineering, 31(6), 466-480.

[P132] McLaughlin, L. (2004). Automated bug tracking: the promise and the pitfalls. IEEE Software, 21(1), 100-103.

[P133] Liu, H., Liu, Q., Niu, Z., & Liu, Y. (2015). Dynamic and automatic feedback-based threshold adaptation for code smell detection. IEEE Transactions on Software Engineering, 42(6), 544-558.

[P134] Dashevskyi, S., Brucker, A. D., & Massacci, F. (2018). A screening test for disclosed vulnerabilities in foss components. IEEE Transactions on Software Engineering, 45(10), 945-966.

[P135] Galster, M., Weyns, D., Tofan, D., Michalik, B., & Avgeriou, P. (2013). Variability in software systems—a systematic literature review. IEEE Transactions on Software Engineering, 40(3), 282-306.

[P136] Moha, N., Guéhéneuc, Y. G., Duchien, L., & Le Meur, A. F. (2009). Decor: A method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering, 36(1), 20-36.

[P137] Liu, H., Guo, X., & Shao, W. (2013). Monitor-based instant software refactoring. IEEE Transactions on Software Engineering, 39(8), 1112-1126.

[P138] McBurney, P. W., Jiang, S., Kessentini, M., Kraft, N. A., Armaly, A., Mkaouer, M. W., & McMillan, C. (2017). Towards prioritizing documentation effort. IEEE Transactions on Software Engineering, 44(9), 897-913.

[P139] da Silva Maldonado, E., Shihab, E., & Tsantalis, N. (2017). Using natural language processing to automatically detect self-admitted technical debt. IEEE Transactions on Software Engineering, 43(11), 1044-1062.

[P140] Guéhéneuc, Y. G., & Antoniol, G. (2008). Demima: A multilayered approach for design pattern identification. IEEE transactions on software engineering, 34(5), 667-684.

[P141] Ye, X., Bunescu, R., & Liu, C. (2015). Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Transactions on Software Engineering, 42(4), 379-402.

[P142] Yan, M., Xia, X., Shihab, E., Lo, D., Yin, J., & Yang, X. (2018). Automating change-level self-admitted technical debt determination. IEEE Transactions on Software Engineering, 45(12), 1211-1229.

[P143] Arnaoudova, V., Eshkevari, L. M., Di Penta, M., Oliveto, R., Antoniol, G., & Guéhéneuc, Y. G. (2014). Repent: Analyzing the nature of identifier renamings. IEEE Transactions on Software Engineering, 40(5), 502-532.

[P144] Duran-Limon, H. A., Garcia-Rios, C. A., Castillo-Barrera, F. E., & Capilla, R. (2015). An ontology-based product architecture derivation approach. IEEE Transactions on Software Engineering, 41(12), 1153-1168.

[P145] Canal, C., Poizat, P., & Salaün, G. (2008). Model-based adaptation of behavioral mismatching components. IEEE Transactions on Software Engineering, 34(4), 546-563.

[P146] Athanasiou, D., Nugroho, A., Visser, J., & Zaidman, A. (2014). Test code quality and its relation to issue handling performance. IEEE Transactions on Software Engineering, 40(11), 1100-1125.

[P147] Ali, N., Guéhéneuc, Y. G., & Antoniol, G. (2012). Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. IEEE Transactions on Software Engineering, 39(5), 725-741.

[P148] Brosch, F., Koziolek, H., Buhnova, B., & Reussner, R. (2011). Architecture-based reliability prediction with the palladio component model. IEEE Transactions on Software Engineering, 38(6), 1319-1339.

[P149] de Jonge, M. (2005). Build-level components. IEEE Transactions on Software Engineering, 31(7), 588-600.

[P150] Sakti, A., Pesant, G., & Guéhéneuc, Y. G. (2014). Instance generator and problem representation to improve object oriented code coverage. IEEE Transactions on Software Engineering, 41(3), 294-313.

[P151] Tsantalis, N., Mazinanian, D., & Krishnan, G. P. (2015). Assessing the refactorability of software clones. IEEE Transactions on Software Engineering, 41(11), 1055-1090.

[P152] Fowkes, J., Chanthirasegaran, P., Ranca, R., Allamanis, M., Lapata, M., & Sutton, C. (2017). Autofolding for source code summarization. IEEE Transactions on Software Engineering, 43(12), 1095-1109.

[P153] Egyed, A. (2010). Automatically detecting and tracking inconsistencies in software design models. IEEE Transactions on Software Engineering, 37(2), 188-204.

[P154] Kästner, C., Dreiling, A., & Ostermann, K. (2013). Variability mining: Consistent semi-automatic detection of product-line features. IEEE Transactions on Software Engineering, 40(1), 67-82.

[P155] Stewart, D. B., & Arora, G. (2003). A tool for analyzing and fine tuning the real-time properties of an embedded system. IEEE Transactions on Software Engineering, 29(4), 311-326.

[P156] Porter, A., Yilmaz, C., Memon, A. M., Schmidt, D. C., & Natarajan, B. (2007). Skoll: A process and infrastructure for distributed continuous quality assurance. IEEE Transactions on Software Engineering, 33(8), 510-525.

[P157] Emberson, P., & Bate, I. (2009). Stressing search with scenarios for flexible solutions to real-time task allocation problems. IEEE Transactions on Software Engineering, 36(5), 704-718.

[P158] Lin, Z., Zhang, X., & Xu, D. (2009). Reverse engineering input syntactic structure from program execution and its applications. IEEE Transactions on Software Engineering, 36(5), 688-703.

[P159] Lavezzoli, P. (2006). The dawn of Indian music in the West. A&C Black.

[P160] Prechelt, L., & Unger, B. (2001). An experiment measuring the effects of personal software process (PSP) training. IEEE transactions on software engineering, 27(5), 465-472.

[P161] Dorn, J., Lacomis, J., Weimer, W., & Forrest, S. (2017). Automatically exploring tradeoffs between software output fidelity and energy costs. IEEE Transactions on Software Engineering, 45(3), 219-236.

[P162] Do, H., Mirarab, S., Tahvildari, L., & Rothermel, G. (2010). The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Transactions on Software Engineering, 36(5), 593-617.

[P163] Chen, N., & Kim, S. (2014). Star: Stack trace based automatic crash reproduction via symbolic execution. IEEE transactions on software engineering, 41(2), 198-220.

[P164] Han, A. R., & Cha, S. (2017). Two-phase assessment approach to improve the efficiency of refactoring identification. IEEE Transactions on Software Engineering, 44(10), 1001-1023.

[P165] Nguyen, B. N., & Memon, A. M. (2014). An observe-model-exercise* paradigm to test event-driven systems with undetermined input spaces. IEEE Transactions on Software Engineering, 40(3), 216-234.

[P166] Dagenais, B., & Robillard, M. P. (2014). Using traceability links to recommend adaptive changes for documentation evolution. IEEE Transactions on Software Engineering, 40(11), 1126-1146.

[P167] Tsantalis, N., & Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. IEEE Transactions on Software Engineering, 35(3), 347-367.

[P168] Fan, Y., Xia, X., Lo, D., & Hassan, A. E. (2018). Chaff from the wheat: Characterizing and determining valid bug reports. IEEE transactions on software engineering, 46(5), 495-525.

[P169] Liu, Y., Xu, C., Cheung, S. C., & Lü, J. (2014). Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. IEEE Transactions on Software Engineering, 40(9), 911-940.

[P170] Wang, X., Dang, Y., Zhang, L., Zhang, D., Lan, E., & Mei, H. (2014). Predicting consistency-maintenance requirement of code clonesat copy-and-paste time. IEEE Transactions on Software Engineering, 40(8), 773-794.

[P171] McMinn, P., Harman, M., Lakhotia, K., Hassoun, Y., & Wegener, J. (2011). Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation. IEEE Transactions on Software Engineering, 38(2), 453-477.

[P172] Lee, S., Wu, R., Cheung, S. C., & Kang, S. (2019). Automatic detection and update suggestion for outdated api names in documentation. IEEE Transactions on Software Engineering, 47(4), 653-675.

[P173] Ghiotto, G., Murta, L., Barros, M., & Van Der Hoek, A. (2018). On the nature of merge conflicts: A study of 2,731 open source Java projects hosted by GitHub. IEEE Transactions on Software Engineering, 46(8), 892-915.

[P174] Xuan, J., Martinez, M., Demarco, F., Clement, M., Marcote, S. L., Durieux, T., ... & Monperrus, M. (2016). Nopol: Automatic repair of conditional statement bugs in java programs. IEEE Transactions on Software Engineering, 43(1), 34-55.

[P175] Carr, S. A., Logozzo, F., & Payer, M. (2016). Automatic contract insertion with ccbot. IEEE Transactions on Software Engineering, 43(8), 701-714.

[P176] Wen, M., Wu, R., & Cheung, S. C. (2018). How well do change sequences predict defects? sequence learning from software changes. IEEE Transactions on Software Engineering, 46(11), 1155-1175.

[P177] Brun, Y., young Bang, J., Edwards, G., & Medvidovic, N. (2015). Self-adapting reliability in distributed software systems. IEEE Transactions on Software Engineering, 41(8), 764-780.

[P178] Meng, Y., Gay, G., & Whalen, M. (2018). Ensuring the observability of structural test obligations. IEEE Transactions on Software Engineering, 46(7), 748-772.

[P179] Fraser, G., & Zeller, A. (2011). Mutation-driven generation of unit tests and oracles. IEEE Transactions on Software Engineering, 38(2), 278-292.

[P180] Gallaba, K., & McIntosh, S. (2018). Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci. IEEE Transactions on Software Engineering, 46(1), 33-50.

[P181] Wang, Y., Yu, H., Zhu, Z., Zhang, W., & Zhao, Y. (2017). Automatic software refactoring via weighted clustering in method-level networks. IEEE Transactions on Software Engineering, 44(3), 202-236.

[P182] Mirakhorli, M., & Cleland-Huang, J. (2015). Detecting, tracing, and monitoring architectural tactics in code. IEEE Transactions on Software Engineering, 42(3), 205-220.

[P183] Chen, T. H., Shang, W., Jiang, Z. M., Hassan, A. E., Nasser, M., & Flora, P. (2016). Finding and evaluating the performance impact of redundant data access for applications that are developed using object-relational mapping frameworks. IEEE Transactions on Software Engineering, 42(12), 1148-1161.

[P184] Jiang, Y., Liu, H., Zhu, J., & Zhang, L. (2018). Automatic and accurate expansion of abbreviations in parameters. IEEE Transactions on Software Engineering, 46(7), 732-747.

[P185] Delahaye, B., Katoen, J. P., Larsen, K. G., Legay, A., Pedersen, M. L., Sher, F., & Wasowski, A. (2011, June). New results on abstract probabilistic automata. In 2011 Eleventh International Conference on Application of Concurrency to System Design (pp. 118-127). IEEE.

[P186] Hrischuk, C. E., & Woodside, C. M. (2002). Logical clock requirements for reverse engineering scenarios from a distributed system. IEEE Transactions on Software Engineering, 28(4), 321-339.

[P187] Desmet, L., Verbaeten, P., Joosen, W., & Piessens, F. (2008). Provable protection against web application vulnerabilities related to session data dependencies. IEEE transactions on software engineering, 34(1), 50-64.

[P188] Bagheri, H., Tang, C., & Sullivan, K. (2016). Automated synthesis and dynamic analysis of tradeoff spaces for object-relational mapping. IEEE Transactions on Software Engineering, 43(2), 145-163.

[P189] Chavez, H. M., Shen, W., France, R. B., Mechling, B. A., & Li, G. (2015). An approach to checking consistency between UML class model and its Java implementation. IEEE Transactions on software engineering, 42(4), 322-344.

[P190] Reder, A., & Egyed, A. (2013). Determining the cause of a design model inconsistency. IEEE Transactions on Software Engineering, 39(11), 1531-1548.

[P191] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., & Li, S. (2017). Measuring program comprehension: A large-scale field study with professionals. IEEE Transactions on Software Engineering, 44(10), 951-976.

[P192] Buse, R. P., & Weimer, W. R. (2009). Learning a metric for code readability. IEEE Transactions on software engineering, 36(4), 546-558.

[P193] Mäder, P., Kuschke, T., & Janke, M. (2019). Reactive auto-completion of modeling activities. IEEE Transactions on Software Engineering, 47(7), 1431-1451.

[P194] Itzik, N., Reinhartz-Berger, I., & Wand, Y. (2015). Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. IEEE Transactions on Software Engineering, 42(7), 687-706.

[P195] Bate, I., Burns, A., & Davis, R. I. (2016). An enhanced bailout protocol for mixed criticality embedded software. IEEE Transactions on Software Engineering, 43(4), 298-320.

[P196] Bennaceur, A., & Issarny, V. (2014). Automated synthesis of mediators to support component interoperability. IEEE Transactions on Software Engineering, 41(3), 221-240.

[P197] Bontemps, Y., Heymans, P., & Schobbens, P. Y. (2005). From live sequence charts to state machines and back: A guided tour. IEEE Transactions on Software Engineering, 31(12), 999-1014.

[P198] Nadi, S., Berger, T., Kästner, C., & Czarnecki, K. (2015). Where do configuration constraints stem from? an extraction approach and an empirical study. IEEE Transactions on Software Engineering, 41(8), 820-841.

[P199] Ravindran, B. (2002). Engineering dynamic real-time distributed systems: Architecture, system description language, and middleware. IEEE Transactions on Software Engineering, 28(1), 30-57.

[P200] Le Goues, C., & Weimer, W. (2011). Measuring code quality to improve specification mining. IEEE Transactions on Software Engineering, 38(1), 175-190.

[P201] Cunha, J., Fernandes, J. P., Mendes, J., & Saraiva, J. (2014). Embedding, evolution, and validation of model-driven spreadsheets. IEEE Transactions on Software Engineering, 41(3), 241-263.

[P202] Chalin, P. (2009). Engineering a sound assertion semantics for the verifying compiler. IEEE Transactions on Software Engineering, 36(2), 275-287.

[P203] Grano, G., Palomba, F., & Gall, H. C. (2019). Lightweight assessment of test-case effectiveness using source-code-quality indicators. IEEE Transactions on Software Engineering, 47(4), 758-774.

[P204] Zuberi, K. M., Pillai, P., & Shin, K. G. (1999, December). EMERALDS: a small-memory real-time microkernel. In Proceedings of the seventeenth ACM symposium on Operating systems principles (pp. 277-299).

[P205] Haran, M., Karr, A., Last, M., Orso, A., Porter, A. A., Sanil, A., & Fouche, S. (2007). Techniques for classifying executions of deployed software to support software engineering tasks. IEEE Transactions on Software Engineering, 33(5), 287-304.

[P206] Ramírez-Mora, S. L., Oktaba, H., Gómez-Adorno, H., & Sierra, G. (2021). Exploring the communication functions of comments during bug fixing in Open Source Software projects. Information and Software Technology, 136, 106584.

[P207] Nam, J., Wang, S., Xi, Y., & Tan, L. (2019). A bug finder refined by a large set of open-source projects. Information and Software Technology, 112, 164-175.

[P208] Lakhotia, K., Harman, M., & Gross, H. (2013). AUSTIN: An open source tool for search based software testing of C programs. Information and Software Technology, 55(1), 112-125.

[P209] Kawaguchi, S., Garg, P. K., Matsushita, M., & Inoue, K. (2006). Mudablue: An automatic categorization system for open source repositories. Journal of Systems and Software, 79(7), 939-953.

[P210] Saidani, I., Ouni, A., Mkaouer, M. W., & Palomba, F. (2021). On the impact of Continuous Integration on refactoring practice: An exploratory study on TravisTorrent. Information and Software Technology, 138, 106618.

[P211] Liu, B., Rong, G., Dong, L., Zhang, H., Chen, D., Chen, T., ... & Zhang, T. (2019). What are the factors affecting the handover process in open source development?. Journal of Systems and Software, 153, 238-254.

[P212] Garousi, V., Keleş, A. B., Balaman, Y., Güler, Z. Ö., & Arcuri, A. (2021). Model-based testing in practice: An experience report from the web applications domain. Journal of Systems and Software, 180, 111032.

[P213] Bian, Y., Koru, G., Su, X., & Ma, P. (2013). SPAPE: A semantic-preserving amorphous procedure extraction method for near-miss clones. Journal of Systems and Software, 86(8), 2077-2093.

[P214] Marcilio, D., Furia, C. A., Bonifácio, R., & Pinto, G. (2020). SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings. Journal of Systems and Software, 168, 110671.

[P215] Sülün, E., Tüzün, E., & Doğrusöz, U. (2021). RSTrace+: Reviewer suggestion using software artifact traceability graphs. Information and Software Technology, 130, 106455.

[P216] Iammarino, M., Zampetti, F., Aversano, L., & Di Penta, M. (2021). An empirical study on the co-occurrence between refactoring actions and self-admitted technical debt removal. Journal of Systems and Software, 178, 110976.

[P217] Hassler, E. E., Hale, D. P., & Hale, J. E. (2018). A comparison of automated training-by-example selection algorithms for Evidence Based Software Engineering. Information and Software Technology, 98, 59-73.

[P218] Wang, D., Kula, R. G., Ishio, T., & Matsumoto, K. (2021). Automatic patch linkage detection in code review using textual content and file location features. Information and Software Technology, 139, 106637.

[P219] Tsai, J. J., & Liu, A. (2009). Experience on knowledge-based software engineering: A logic-based requirements language and its industrial applications. Journal of Systems and Software, 82(10), 1578-1587.

[P220] Lizcano, D., Soriano, J., López, G., & Gutiérrez, J. J. (2017). Automatic verification and validation wizard in web-centred end-user software engineering. Journal of Systems and Software, 125, 47-67.

[P221] Golzadeh, M., Decan, A., Legay, D., & Mens, T. (2021). A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. Journal of Systems and Software, 175, 110911.

[P222] Cordy, J. R., Dean, T. R., Malton, A. J., & Schneider, K. A. (2002). Source transformation in software engineering using the TXL transformation system. Information and Software Technology, 44(13), 827-837.

[P223] Salman, H. E. (2021). Feature-based insight for forks in social coding platforms. Information and Software Technology, 140, 106679.

[P224] Moseler, O., Lemmer, F., Baltes, S., & Diehl, S. (2021). On the diversity and frequency of code related to mathematical formulas in real-world Java projects. Journal of Systems and Software, 172, 110863.

[P225] Grundy, J., Mugridge, W., & Hosking, J. (2000). Constructing component-based software engineering environments: issues and experiences. Information and Software Technology, 42(2), 103-114.

[P226] Magdaleno, A. M., Werner, C. M. L., & De Araujo, R. M. (2012). Reconciling software development models: A quasi-systematic review. Journal of Systems and Software, 85(2), 351-369.

[P227] Li, C., Huang, L., Ge, J., Luo, B., & Ng, V. (2018). Automatically classifying user requests in crowdsourcing requirements engineering. Journal of Systems and Software, 138, 108-123.

[P228] Saied, M. A., Ouni, A., Sahraoui, H., Kula, R. G., Inoue, K., & Lo, D. (2018). Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software, 145, 164-179.

[P229] de Lima Júnior, M. L., Soares, D. M., Plastino, A., & Murta, L. (2018). Automatic assignment of integrators to pull requests: The importance of selecting appropriate attributes. Journal of Systems and Software, 144, 181-196.

[P230] Auch, M., Weber, M., Mandl, P., & Wolff, C. (2020). Similarity-based analyses on software applications: A systematic literature review. Journal of Systems and Software, 168, 110669.

[P231] Chong, C. Y., & Lee, S. P. (2017). Automatic clustering constraints derivation from object-oriented software using weighted complex network with graph theory analysis. Journal of Systems and Software, 133, 28-53.

[P232] Rahman, A., & Williams, L. (2019). Source code properties of defective infrastructure as code scripts. Information and Software Technology, 112, 148-163.

[P233] Montandon, J. E., Valente, M. T., & Silva, L. L. (2021). Mining the technical roles of github users. Information and Software Technology, 131, 106485.

[P234] Miryeganeh, N., Hashtroudi, S., & Hemmati, H. (2021). GloBug: Using global data in Fault Localization. Journal of Systems and Software, 177, 110961.

[P235] Wang, S., Bansal, C., & Nagappan, N. (2021). Large-scale intent analysis for identifying large-review-effort code changes. Information and Software Technology, 130, 106408.

[P236] Meqdadi, O., Alhindawi, N., Alsakran, J., Saifan, A., & Migdadi, H. (2019). Mining software repositories for adaptive change commits using machine learning techniques. Information and Software Technology, 109, 80-91.

[P237] Fu, Y., Yan, M., Zhang, X., Xu, L., Yang, D., & Kymer, J. D. (2015). Automated classification of software change messages by semi-supervised latent dirichlet allocation. Information and Software Technology, 57, 369-377.

[P238] Newman, C. D., AlSuhaibani, R. S., Decker, M. J., Peruma, A., Kaushik, D., Mkaouer, M. W., & Hill, E. (2020). On the generation, structure, and semantics of grammar patterns in source code identifiers. Journal of Systems and Software, 170, 110740.

[P239] Guerra, E., Alves, F., Kulesza, U., & Fernandes, C. (2013). A reference architecture for organizing the internal structure of metadata-based frameworks. Journal of Systems and Software, 86(5), 1239-1256.

[P240] Yan, M., Fu, Y., Zhang, X., Yang, D., Xu, L., & Kymer, J. D. (2016). Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project. Journal of Systems and Software, 113, 296-308.

[P241] Sarkar, S., Maskeri, G., & Ramachandran, S. (2009). Discovery of architectural layers and measurement of layering violations in source code. Journal of Systems and Software, 82(11), 1891-1905.

[P242] Lima, P., Guerra, E., Meirelles, P., Kanashiro, L., Silva, H., & Silveira, F. F. (2018). A metrics suite for code annotation assessment. Journal of Systems and Software, 137, 163-183.

[P243] Mo, R., & Yin, Z. (2020). Exploring software bug-proneness based on evolutionary clique modeling and analysis. Information and Software Technology, 128, 106380.

[P244] Coppola, R., Ardito, L., Torchiano, M., & Alégroth, E. (2021). Translation from layout-based to visual android test scripts: An empirical evaluation. Journal of Systems and Software, 171, 110845.

[P245] Qiu, D., Li, B., & Leung, H. (2016). Understanding the API usage in Java. Information and software technology, 73, 81-100.

[P246] Xiao, X., Pan, Y., Zhang, B., Hu, G., Li, Q., & Lu, R. (2021). ALBFL: A novel neural ranking model for software fault localization via combining static and dynamic features. Information and Software Technology, 139, 106653.

[P247] Lakhotia, K., McMinn, P., & Harman, M. (2010). An empirical investigation into branch coverage for C programs using CUTE and AUSTIN. Journal of Systems and Software, 83(12), 2379-2391.

[P248] Pradhan, D., Wang, S., Yue, T., Ali, S., & Liaaen, M. (2019). Search-based test case implantation for testing untested configurations. Information and Software Technology, 111, 22-36.

[P249] Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., & Cukic, B. (2017). Automated triaging of very large bug repositories. Information and software technology, 89, 1-13.

[P250] Shippey, T., Bowes, D., & Hall, T. (2019). Automatically identifying code features for software defect prediction: Using AST N-grams. Information and Software Technology, 106, 142-160.

[P251] Şora, I., & Chirila, C. B. (2019). Finding key classes in object-oriented software systems by techniques based on static analysis. Information and Software Technology, 116, 106176.

[P252] Huang, Y., Hu, X., Jia, N., Chen, X., Zheng, Z., & Luo, X. (2020). CommtPst: Deep learning source code for commenting positions prediction. Journal of Systems and Software, 170, 110754.

[P253] Yan, M., Zhang, X., Liu, C., Xu, L., Yang, M., & Yang, D. (2017). Automated change-prone class prediction on unlabeled dataset using unsupervised method. Information and Software Technology, 92, 1-16.

[P254] Gu, Y., Xuan, J., Zhang, H., Zhang, L., Fan, Q., Xie, X., & Qian, T. (2019). Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence. Journal of Systems and Software, 148, 88-104.

[P255] Malek, S., Krishnan, H. R., & Srinivasan, J. (2010). Enhancing middleware support for architecture-based development through compositional weaving of styles. Journal of Systems and Software, 83(12), 2513-2527.

[P256] Li, Z., Jiang, Z., Chen, X., Cao, K., & Gu, Q. (2021). Laprob: A label propagation-based software bug localization method. Information and Software Technology, 130, 106410.

[P257] Lemos, O. A. L., Bajracharya, S., Ossher, J., Masiero, P. C., & Lopes, C. (2011). A test-driven approach to code search and its application to the reuse of auxiliary functionality. Information and Software Technology, 53(4), 294-306.

[P258] Mazhelis, O., Tyrväinen, P., & Frank, L. (2013). Vertical software industry evolution: The impact of software costs and limited customer base. Information and Software Technology, 55(4), 690-698.

[P259] Ras, E., & Rech, J. (2009). Using Wikis to support the Net Generation in improving knowledge acquisition in capstone projects. Journal of Systems and Software, 82(4), 553-562.

[P260] Yu, K., Lin, M., Chen, J., & Zhang, X. (2012). Towards automated debugging in software evolution: Evaluating delta debugging on real regression bugs from the developers' perspectives. Journal of Systems and Software, 85(10), 2305-2317.

[P261] Zhu, L., Bui, N. B., Liu, Y., & Gorton, I. (2007). MDABench: Customized benchmark generation using MDA. Journal of Systems and Software, 80(2), 265-282.

[P262] Haitzer, T., Navarro, E., & Zdun, U. (2017). Reconciling software architecture and source code in support of software evolution. Journal of Systems and Software, 123, 119-144.

[P263] Chen, Z., Guo, H. F., & Song, M. (2018). Improving regression test efficiency with an awareness of refactoring changes. Information and Software Technology, 103, 174-187.

[P264] Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., ... & Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 157, 110380.

[P265] Haitzer, T., & Zdun, U. (2015). Semi-automatic architectural pattern identification and documentation using architectural primitives. Journal of Systems and Software, 102, 35-57.

[P266] Pill, I., & Wotawa, F. (2018). Automated generation of (F) LTL oracles for testing and debugging. Journal of Systems and Software, 139, 124-141.

[P267] Eler, M. M., Endo, A. T., & Durelli, V. H. (2016). An empirical study to quantify the characteristics of Java programs that may influence symbolic execution from a unit testing perspective. Journal of Systems and Software, 121, 281-297.

[P268] Xiaobo, Y., Bin, L., Shihai, W., Dong, A., Feng, Z., & Yelin, Y. (2021). Efilter: An effective fault localization based on information entropy with unlabelled test cases. Information and Software Technology, 134, 106543.

[P269] Schmidt, H. (2003). Trustworthy components—compositionality and prediction. Journal of Systems and Software, 65(3), 215-225.

[P270] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., & Aho, P. (2010). Knowledge based quality-driven architecture design and evaluation. Information and Software Technology, 52(6), 577-601.

[P271] Liu, K., Li, L., Koyuncu, A., Kim, D., Liu, Z., Klein, J., & Bissyandé, T. F. (2021). A critical review on the evaluation of automated program repair systems. Journal of Systems and Software, 171, 110817.

[P272] Xuan, J., Cornu, B., Martinez, M., Baudry, B., Seinturier, L., & Monperrus, M. (2016). B-Refactoring: Automatic test code refactoring to improve dynamic analysis. Information and Software Technology, 76, 65-80.

[P273] Linsbauer, L., Schwägerl, F., Berger, T., & Grünbacher, P. (2021). Concepts of variation control systems. Journal of Systems and Software, 171, 110796.

[P274] Erdemir, U., & Buzluca, F. (2014). A learning-based module extraction method for object-oriented systems. Journal of Systems and Software, 97, 156-177.

[P275] Kochovski, P., Drobintsev, P. D., & Stankovski, V. (2019). Formal quality of service assurances, ranking and verification of cloud deployment options with a probabilistic model checking method. Information and Software Technology, 109, 14-25.

[P276] Ruiz-Rube, I., Dodero, J. M., & Colomo-Palacios, R. (2015). A framework for software process deployment and evaluation. Information and Software Technology, 59, 205-221.

[P277] Al Dallal, J., & Briand, L. C. (2010). An object-oriented high-level design-based class cohesion metric. Information and software technology, 52(12), 1346-1361.

[P278] Vujošević-Janičić, M., Nikolić, M., Tošić, D., & Kuncak, V. (2013). Software verification and graph similarity for automated evaluation of students' assignments. Information and Software Technology, 55(6), 1004-1016.

[P279] Dietrich, R., Schmitt, F., Grund, A., & Stolle, J. (2017). Critical-blame analysis for OpenMP 4.0 offloading on Intel Xeon Phi. Journal of Systems and Software, 125, 381-388.

[P280] Chong, C. Y., Lee, S. P., & Ling, T. C. (2013). Efficient software clustering technique using an adaptive and preventive dendrogram cutting approach. Information and Software Technology, 55(11), 1994-2012.

[P281] Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2015). A time-based approach to automatic bug report assignment. Journal of Systems and Software, 102, 109-122.

[P282] Goseva-Popstojanova, K., & Perhinschi, A. (2015). On the capability of static code analysis to detect security vulnerabilities. Information and Software Technology, 68, 18-33.

[P283] Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software, 117, 166-184.

[P284] Ruiz, F. J. B., Ramón, Ó. S., & Molina, J. G. (2017). A tool to support the definition and enactment of model-driven migration processes. Journal of Systems and Software, 128, 106-129.

[P285] Lukins, S. K., Kraft, N. A., & Etzkorn, L. H. (2010). Bug localization using latent dirichlet allocation. Information and Software Technology, 52(9), 972-990.

[P286] Chan, W. K., Cheung, S. C., Ho, J. C., & Tse, T. H. (2009). PAT: a pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. Journal of Systems and Software, 82(3), 422-434.

[P287] Kessentini, W., Sahraoui, H., & Wimmer, M. (2019). Automated metamodel/model co-evolution: A search-based approach. Information and Software Technology, 106, 49-67.

[P288] Wohlrab, R., Knauss, E., & Pelliccione, P. (2020). Why and how to balance alignment and diversity of requirements engineering practices in automotive. Journal of Systems and Software, 162, 110516.

[P289] Al Dallal, J. (2012). Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics. Information and Software Technology, 54(10), 1125-1141.

[P290] Cirilo, E., Nunes, I., Kulesza, U., & Lucena, C. (2012). Automating the product derivation process of multi-agent systems product lines. Journal of Systems and Software, 85(2), 258-276.

[P291] Feldmann, S., Kernschmidt, K., Wimmer, M., & Vogel-Heuser, B. (2019). Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering. Journal of Systems and Software, 153, 105-134.

[P292] Eichelberger, H., & Schmid, K. (2009). Guidelines on the aesthetic quality of UML class diagrams. Information and Software Technology, 51(12), 1686-1698.

[P293] Nadal, S., Herrero, V., Romero, O., Abelló, A., Franch, X., Vansummeren, S., & Valerio, D. (2017). A software reference architecture for semantic-aware Big Data systems. Information and software technology, 90, 75-92.

[P294] Teixeira, S., Agrizzi, B. A., Pereira Filho, J. G., Rossetto, S., Pereira, I. S. A., Costa, P. D., ... & Martinelli, R. R. (2020). LAURA architecture: Towards a simpler way of building situation-aware and business-aware IoT applications. Journal of Systems and Software, 161, 110494.

[P295] O'Keeffe, M., & Cinnéide, M. O. (2008). Search-based refactoring for software maintenance. Journal of Systems and Software, 81(4), 502-516.

[P296] Kuttal, S. K., Sarma, A., Rothermel, G., & Wang, Z. (2018). What happened to my application? Helping end users comprehend evolution through variation management. Information and Software Technology, 103, 55-74.

[P297] Blouin, A., Lelli, V., Baudry, B., & Coulon, F. (2018). User interface design smell: Automatic detection and refactoring of Blob listeners. Information and Software Technology, 102, 49-64.

[P298] Abdullah, I. S., & Menascé, D. A. (2013). The Meta-Protocol framework. Journal of Systems and Software, 86(11), 2711-2724.

[P299] Al-Hroob, A., Imam, A. T., & Al-Heisa, R. (2018). The use of artificial neural networks for extracting actions and actors from requirements document. Information and Software Technology, 101, 1-15.

[P300] Han, A. R., Bae, D. H., & Cha, S. (2015). An efficient approach to identify multiple and independent move method refactoring candidates. Information and Software Technology, 59, 53-66.

[P301] Vogel-Heuser, B., Fay, A., Schaefer, I., & Tichy, M. (2015). Evolution of software in automated production systems: Challenges and research directions. Journal of Systems and Software, 110, 54-84.

[P302] Cortellessa, V., & Di Pompeo, D. (2021). Analyzing the sensitivity of multi-objective software architecture refactoring to configuration characteristics. Information and Software Technology, 135, 106568.

[P303] Magalhaes, A. P. F., Andrade, A. M. S., & Maciel, R. S. P. (2019). Model driven transformation development (MDTD): An approach for developing model to model transformation. Information and Software Technology, 114, 55-76.

[P304] Shafiee, S., Wautelet, Y., Hvam, L., Sandrin, E., & Forza, C. (2020). Scrum versus Rational Unified Process in facing the main challenges of product configuration systems development. Journal of Systems and Software, 170, 110732.

[P305] Boucher, A., & Badri, M. (2018). Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. Information and Software Technology, 96, 38-67.

[P306] Pira, E., Rafe, V., & Nikanjam, A. (2018). Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations. Information and Software Technology, 97, 110-134.

[P307] Boes, J., & Migeon, F. (2017). Self-organizing multi-agent systems for the control of complex systems. Journal of Systems and Software, 134, 12-28.

[P308] Heinrich, R., Koch, S., Cha, S., Busch, K., Reussner, R., & Vogel-Heuser, B. (2018). Architecture-based change impact analysis in cross-disciplinary automated production systems. Journal of Systems and Software, 146, 167-185.

[P309] Smidts, C., Huang, X., & Widmaier, J. C. (2002). Producing reliable software: an experiment. Journal of Systems and Software, 61(3), 213-224.

[P310] Lin, Q., Low, C. P., Ng, J. M., Bu, J., & Liu, X. (2003). Multiuser collaborative work in virtual environment based CASE tool. Information and Software Technology, 45(5), 253-267.

[P311] Butting, A., Kautz, O., Rumpe, B., & Wortmann, A. (2019). Continuously analyzing finite, message-driven, time-synchronous component & connector systems during architecture evolution. Journal of Systems and Software, 149, 437-461.

[P312] Han, A. R., & Bae, D. H. (2013). Dynamic profiling-based approach to identifying cost-effective refactorings. Information and Software Technology, 55(6), 966-985.

[P313] Ferrer, J., Chicano, F., & Alba, E. (2013). Estimating software testing complexity. Information and Software Technology, 55(12), 2125-2139.

[P314] Alkhazi, B., Abid, C., Kessentini, M., & Wimmer, M. (2020). On the value of quality attributes for refactoring ATL model transformations: A multi-objective approach. Information and Software Technology, 120, 106243.

[P315] Autili, M., Mostarda, L., Navarra, A., & Tivoli, M. (2008). Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems. Journal of Systems and Software, 81(12), 2210-2236.

[P316] Staron, M., & Meding, W. (2016). Mesram–a method for assessing robustness of measurement programs in large software development organizations and its industrial evaluation. Journal of Systems and Software, 113, 76-100.

[P317] Bellettini, C., Damiani, E., & Fugini, M. G. (2001). Software reuse in-the-small: automating group rewarding. Information and Software Technology, 43(11), 651-660.

[P318] Abushark, Y., Thangarajah, J., Harland, J., & Miller, T. (2017). A framework for automatically ensuring the conformance of agent designs. Journal of Systems and Software, 131, 266-310.

[P319] Vogel-Heuser, B., Fischer, J., Feldmann, S., Ulewicz, S., & Rösch, S. (2017). Modularity and architecture of PLC-based software for automated production Systems: An analysis in industrial companies. Journal of Systems and Software, 131, 35-62.

[P320] Edagawa, T. A. Y. S. S. T. T., Akaike, T., Higo, Y., Kusumoto, S., Hanabusa, S., & Shibamoto, T. (2011). Function point measurement from Web application source code based on screen transitions and database accesses. Journal of Systems and Software, 84(6), 976-984.

[P321] Bracciali, A., Brogi, A., & Canal, C. (2005). A formal approach to component adaptation. Journal of Systems and Software, 74(1), 45-54.

[P322] Izquierdo, J. L. C., Jouault, F., Cabot, J., & Molina, J. G. (2012). API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. Information and Software Technology, 54(3), 257-273.

[P323] Delgado-Perez, P., & Chicano, F. (2020). An experimental and practical study on the equivalent mutant connection: An evolutionary approach. Information and Software Technology, 124, 106317.

[P324] Guo, J., White, J., Wang, G., Li, J., & Wang, Y. (2011). A genetic algorithm for optimized feature selection with resource constraints in software product lines. Journal of Systems and Software, 84(12), 2208-2221.

[P325] Wood, S., Michaelides, G., & Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking?. Information and Software Technology, 55(4), 660-672.

[P326] Staron, M., Meding, W., & Nilsson, C. (2009). A framework for developing measurement systems and its industrial evaluation. Information and Software Technology, 51(4), 721-737.

[P327] Rebai, S., Kessentini, M., Wang, H., & Maxim, B. (2020). Web service design defects detection: A bi-level multi-objective approach. Information and Software Technology, 121, 106255.

[P328] Parra, E., Dimou, C., Llorens, J., Moreno, V., & Fraga, A. (2015). A methodology for the classification of quality of requirements using machine learning techniques. Information and Software Technology, 67, 180-195.

[P329] Pelliccione, P., Tivoli, M., Bucchiarone, A., & Polini, A. (2008). An architectural approach to the correct and automatic assembly of evolving component-based systems. Journal of Systems and Software, 81(12), 2237-2251.

[P330] Pančur, M., & Ciglarič, M. (2011). Impact of test-driven development on productivity, code and tests: A controlled experiment. Information and Software Technology, 53(6), 557-573.

[P331] Kebir, S., Borne, I., & Meslati, D. (2017). A genetic algorithm-based approach for automated refactoring of component-based software. Information and Software Technology, 88, 17-36.

[P332] Pernstål, J., Gorschek, T., Feldt, R., & Florén, D. (2015). Requirements communication and balancing in large-scale software-intensive product development. Information and Software Technology, 67, 44-64.

[P333] Stamelos, I., Vlahavas, I., Refanidis, I., & Tsoukiàs, A. (2000). Knowledge based evaluation of software systems: a case study. Information and Software Technology, 42(5), 333-345.

[P334] Fortino, G., & Russo, W. (2012). ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems. Information and Software Technology, 54(6), 608-624.

[P335] Spanoudakis, G., Zisman, A., Pérez-Minana, E., & Krause, P. (2004). Rule-based generation of requirements traceability relations. Journal of systems and software, 72(2), 105-127.

[P336] Szőke, Á. (2011). Conceptual scheduling model and optimized release scheduling for agile environments. Information and software technology, 53(6), 574-591.

[P337] Briand, L. C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. Information and Software Technology, 51(1), 16-30.

[P338] Etemaadi, R., Lind, K., Heldal, R., & Chaudron, M. R. (2013). Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system. Journal of Systems and Software, 86(10), 2559-2573.

[P339] Ribeiro, J. C. B., Zenha-Rela, M. A., & de Vega, F. F. (2009). Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software. Information and Software Technology, 51(11), 1534-1548.

[P340] Chang, T. K., & Hwang, G. H. (2007). The design and implementation of an application program interface for securing XML documents. Journal of Systems and Software, 80(8), 1362-1374.

[P341] Pernstål, J., Feldt, R., & Gorschek, T. (2013). The lean gap: A review of lean approaches to large-scale software systems development. Journal of Systems and Software, 86(11), 2797-2821.

[P342] Hotomski, S., & Glinz, M. (2019). GuideGen: An approach for keeping requirements and acceptance tests aligned via automatically generated guidance. Information and Software Technology, 110, 17-38.

[P343] George, B., & Williams, L. (2004). A structured experiment of test-driven development. Information and software Technology, 46(5), 337-342.

[P344] Woo, G., Chae, H. S., Cui, J. F., & Ji, J. H. (2009). Revising cohesion measures by considering the impact of write interactions between class members. Information and Software Technology, 51(2), 405-417.

[P345] Ding, Y., Ray, B., Devanbu, P., & Hellendoorn, V. J. (2020, December). Patching as translation: the data and the metaphor. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (pp. 275-286).

[P346] Li, M., Shi, L., Yang, Y., & Wang, Q. (2020, December). A deep multitask learning approach for requirements discovery and annotation from open forum. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (pp. 336-348).

[P347] Qu, Y., Liu, T., Chi, J., Jin, Y., Cui, D., He, A., & Zheng, Q. (2018, September). node2defect: using network embedding to improve software defect prediction. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 844-849).

[P348] Pinto, G., Canino, A., Castor, F., Xu, G., & Liu, Y. D. (2017, October). Understanding and overcoming parallelism bottlenecks in forkjoin applications. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 765-775). IEEE.

[P349] Yang, Y., Jiang, Y., Gu, M., Sun, J., Gao, J., & Liu, H. (2017, October). A language model for statements of software code. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 682-687). IEEE.

[P350] Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016, August). Usage, costs, and benefits of continuous integration in open-source projects. In Proceedings of the 31st IEEE/ACM international conference on automated software engineering (pp. 426-437).

[P351] Nam, J., & Kim, S. (2015, November). Clami: Defect prediction on unlabeled datasets (t). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 452-463). IEEE.

[P352] Xiao, X., Li, S., Xie, T., & Tillmann, N. (2013, November). Characteristic studies of loop problems for structural test generation via symbolic execution. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 246-256). IEEE.

[P353] Cai, Y., Grundy, J., & Hosking, J. (2004, September). Experiences integrating and scaling a performance test bed generator with an open source case tool. In Proceedings. 19th International Conference on Automated Software Engineering, 2004. (pp. 36-45). IEEE.

[P354] Mayr-Dorn, C., Vierhauser, M., Bichler, S., Keplinger, F., Cleland-Huang, J., Egyed, A., & Mehofer, T. (2021, May). Supporting quality assurance with automated process-centric quality constraints checking. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 1298-1310). IEEE.

[P355] Bui, N. D., Yu, Y., & Jiang, L. (2021, May). Infercode: Self-supervised learning of code representations by predicting subtrees. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 1186-1197). IEEE.

[P356] Zhai, J., Xu, X., Shi, Y., Tao, G., Pan, M., Ma, S., ... & Zhang, X. (2020, June). CPC: Automatically classifying and propagating natural language comments via program analysis. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 1359-1371).

[P357] Liu, K., Wang, S., Koyuncu, A., Kim, K., Bissyandé, T. F., Kim, D., ... & Traon, Y. L. (2020, June). On the efficiency of test suite based program repair: A systematic assessment of 16 automated repair systems for java programs. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 615-627).

[P358] Islam, M. J., Pan, R., Nguyen, G., & Rajan, H. (2020, June). Repairing deep neural networks: Fix patterns and challenges. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 1135-1146).

[P359] Arya, D., Wang, W., Guo, J. L., & Cheng, J. (2019, May). Analysis and detection of information types of open source software issue discussions. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 454-464). IEEE.

[P360] Joblin, M., Mauerer, W., Apel, S., Siegmund, J., & Riehle, D. (2015, May). From developer networks to verified communities: A fine-grained approach. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 563-573). IEEE.

[P361] Zhang, Y., Chen, Z., Wang, J., Dong, W., & Liu, Z. (2015, May). Regular property guided dynamic symbolic execution. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 643-653). IEEE.

[P362] Zanetti, M. S., Scholtes, I., Tessone, C. J., & Schweitzer, F. (2013, May). Categorizing bugs with social networks: a case study on four open source software communities. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 1032-1041). IEEE.

[P363] Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., & Menzies, T. (2013, May). Automatic query reformulations for text retrieval in software engineering. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 842-851). IEEE.

[P364] Babic, D., & Hu, A. J. (2008, May). Calysto: scalable and precise extended static checking. In Proceedings of the 30th international conference on Software engineering (pp. 211-220).

[P365] Kim, M., Notkin, D., & Grossman, D. (2007, May). Automatic inference of structural changes for matching across program versions. In 29th International Conference on Software Engineering (ICSE'07) (pp. 333-343). IEEE.

[P366] Alves, I., & Rocha, C. (2021, May). Qualifying software engineers undergraduates in DevOps-challenges of introducing technical and non-technical concepts in a project-oriented course. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 144-153). IEEE.

[P367] Tan, S. H., Hu, C., Li, Z., Zhang, X., & Zhou, Y. (2021, May). GitHub-OSS Fixit: Fixing Bugs at Scale in a Software Engineering Course. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 1-10). IEEE.

[P368] Perez, B. (2021, May). Enhancing the learning of database access programming using continuous integration and aspect oriented programming. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 221-230). IEEE.

[P369] Angermeir, F., Voggenreiter, M., Moyon, F., & Mendez, D. (2021, May). Enterprise-driven open source software: a case study on security automation. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 278-287). IEEE.

[P370] Li, H., Khomh, F., & Openja, M. (2021, September). Understanding Quantum Software Engineering Challenges An Empirical Study on Stack Exchange Forums and GitHub Issues. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 343-354). IEEE.

[P371] Anderson, J., Steinmacher, I., & Rodeghero, P. (2020, September). Assessing the Characteristics of FOSS Contributions in Network Automation Projects. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 324-335). IEEE.

[P372] Rahimi, M., Goss, W., & Cleland-Huang, J. (2016, October). Evolving requirements-to-code trace links across versions of a software system. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 99-109). IEEE.

[P373] Blaz, C. C. A., & Becker, K. (2016, May). Sentiment analysis in tickets for it support. In Proceedings of the 13th International Conference on Mining Software Repositories (pp. 235-246).

[P374] Maalej, W., & Happel, H. J. (2010, May). Can development work describe itself?. In 2010 7th IEEE working conference on mining software repositories (MSR 2010) (pp. 191-200). IEEE.

[P375] Enslen, E., Hill, E., Pollock, L., & Vijay-Shanker, K. (2009, May). Mining source code to automatically split identifiers for software analysis. In 2009 6th IEEE International Working Conference on Mining Software Repositories (pp. 71-80). IEEE.

[P376] Wang, W., Niu, N., Liu, H., & Niu, Z. (2018, August). Enhancing automated requirements traceability by resolving polysemy. In 2018 IEEE 26th International Requirements Engineering Conference (RE) (pp. 40-51). IEEE.

[P377] Paul, R., Bosu, A., & Sultana, K. Z. (2019, February). Expressions of sentiments during code reviews: Male vs. female. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 26-37). IEEE.

[P378] Alexandru, C. V., Panichella, S., & Gall, H. C. (2017, February). Reducing redundancies in multi-revision code analysis. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 148-159). IEEE.

[P379] Beller, M., Bholanath, R., McIntosh, S., & Zaidman, A. (2016, March). Analyzing the state of static analysis: A large-scale evaluation in open source software. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (Vol. 1, pp. 470-481). IEEE.

[P380] Campbell, J. C., Santos, E. A., & Hindle, A. (2016, May). The unreasonable effectiveness of traditional information retrieval in crash report deduplication. In Proceedings of the 13th International Conference on Mining Software Repositories (pp. 269-280).

[P381] Brown, C., & Parnin, C. (2020, November). Understanding the impact of GitHub suggested changes on recommendations between developers. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1065-1076).

[P382] Erlenhov, L., Neto, F. G. D. O., & Leitner, P. (2020, November). An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (pp. 445-455).

[P383] Nguyen, H. A., Dyer, R., Nguyen, T. N., & Rajan, H. (2014, November). Mining preconditions of APIs in large-scale code corpus. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 166-177).

[P384] Mattis, T., Rein, P., Dursch, F., & Hirschfeld, R. (2020, June). RTPTorrent: An open-source dataset for evaluating regression test prioritization. In Proceedings of the 17th international conference on mining software repositories (pp. 385-396).

[P385] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., & Filkov, V. (2015, August). Quality and productivity outcomes relating to continuous integration in GitHub. In Proceedings of the 2015 10th joint meeting on foundations of software engineering (pp. 805-816).

[P386] Foo, D., Chua, H., Yeo, J., Ang, M. Y., & Sharma, A. (2018, October). Efficient static checking of library updates. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 791-796).

[P387] Yu, Y., Wang, H., Yin, G., & Wang, T. (2016). Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?. Information and Software Technology, 74, 204-218.

[P388] Mukherjee, S., Almanza, A., & Rubio-Gonzalez, C. (2021, July). Fixing dependency errors for Python build reproducibility. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 439-451).

[P389] Shi, L., Chen, X., Yang, Y., Jiang, H., Jiang, Z., Niu, N., & Wang, Q. (2021, August). A first look at developers' live chat on gitter. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 391-403).

[P390] Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., & Lo, D. (2019). Categorizing the content of github readme files. Empirical Software Engineering, 24, 1296-1327.

[P391] Izquierdo, J. L. C., & Cabot, J. (2015, May). Enabling the definition and enforcement of governance rules in open source systems. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 505-514). IEEE.

[P392] Almhana, R., & Kessentini, M. (2021). Considering dependencies between bug reports to improve bugs triage. Automated Software Engineering, 28, 1-26.

[P393] Di Penta, M., German, D. M., Gueheneuc, Y. G., & Antoniol, G. (2010, May). An exploratory study of the evolution of software licensing. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1 (pp. 145-154).

[P394] Zhou, P., Liu, J., Yang, Z., & Zhou, G. (2017, February). Scalable tag recommendation for software information sites. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 272-282). IEEE.

[P395] Urli, S., Yu, Z., Seinturier, L., & Monperrus, M. (2018, May). How to design a program repair bot? insights from the repairnator project. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (pp. 95-104).

[P396] Hasan, M., Iqbal, A., Islam, M. R. U., Rahman, A. I., & Bosu, A. (2021). Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report. Empirical Software Engineering, 26, 1-34.

[P397] Izadi, M., Heydarnoori, A., & Gousios, G. (2021). Topic recommendation for software repositories using multi-label classification algorithms. Empirical Software Engineering, 26, 1-33.

[P398] Syer, M. D., Shang, W., Jiang, Z. M., & Hassan, A. E. (2017). Continuous validation of performance test workloads. Automated Software Engineering, 24, 189-231.

[P399] Shi, L., Jiang, Z., Yang, Y., Chen, X., Zhang, Y., Mu, F., ... & Wang, Q. (2021, November). ISPY: Automatic issue-solution pair extraction from community live chats. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 142-154). IEEE.

[P400] Tuarob, S., Assavakamhaenghan, N., Tanaphantaruk, W., Suwanworaboon, P., Hassan, S. U., & Choetkiertikul, M. (2021). Automatic team recommendation for collaborative software development. Empirical Software Engineering, 26(4), 64.

[P401] Nizamani, Z. A., Liu, H., Chen, D. M., & Niu, Z. (2018). Automatic approval prediction for software enhancement requests. Automated Software Engineering, 25, 347-381.

[P402] Nyamawe, A. S., Liu, H., Niu, N., Umer, Q., & Niu, Z. (2019, September). Automated recommendation of software refactorings based on feature requests. In 2019 IEEE 27th International Requirements Engineering Conference (RE) (pp. 187-198). IEEE.

[P403] Chatterjee, A., Guizani, M., Stevens, C., Emard, J., May, M. E., Burnett, M., & Ahmed, I. (2021, May). AID: An automated detector for gender-inclusivity bugs in OSS project pages. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 1423-1435). IEEE.

[P404] Honsel, D., Herbold, V., Waack, S., & Grabowski, J. (2021). Investigation and prediction of open source software evolution using automated parameter mining for agent-based simulation. Automated Software Engineering, 28(1), 3.

[P405] Imran, M. M., Ciborowska, A., & Damevski, K. (2021, May). Automatically selecting follow-up questions for deficient bug reports. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (pp. 167-178). IEEE.

[P406] Fischer, G. (2008). Rethinking software design in participation cultures. Automated Software Engineering, 15, 365-377.

[P407] Ragkhitwetsagul, C., & Krinke, J. (2019). Siamese: scalable and incremental code clone search via multiple code representations. Empirical Software Engineering, 24(4), 2236-2284.

[P408] Mahajan, S., Abolhassani, N., & Prasad, M. R. (2020, November). Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1052-1064).

[P409] Mirhosseini, S., & Parnin, C. (2017, October). Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In 2017 32nd IEEE/ACM international conference on automated software engineering (ASE) (pp. 84-94). IEEE.

[P410] Liu, H., Sun, C., Su, Z., Jiang, Y., Gu, M., & Sun, J. (2017, May). Stochastic optimization of program obfuscation. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 221-231). IEEE.

[P411] Noda, K., Yokoyama, H., & Kikuchi, S. (2021, September). Sirius: Static program repair with dependence graph-based systematic edit patterns. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 437-447). IEEE.

[P412] Wang, X., Zhang, L., Xie, T., Mei, H., & Sun, J. (2009, May). Locating need-to-translate constant strings for software internationalization. In 2009 IEEE 31st International Conference on Software Engineering (pp. 353-363). IEEE.

[P413] Paixao, M., Krinke, J., Han, D., Ragkhitwetsagul, C., & Harman, M. (2017, October). Are developers aware of the architectural impact of their changes?. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 95-105). IEEE.

[P414] Miraldo, V. C., & Swierstra, W. (2019). An efficient algorithm for type-safe structural diffing. Proceedings of the ACM on Programming Languages, 3(ICFP), 1-29.

[P415] Wong, E., Yang, J., & Tan, L. (2013, November). Autocomment: Mining question and answer sites for automatic comment generation. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 562-567). IEEE.

[P416] Lu, Y., Chaudhuri, S., Jermaine, C., & Melski, D. (2018, May). Program splicing. In Proceedings of the 40th International Conference on Software Engineering (pp. 338-349).

[P417] Messaoudi, S., Shin, D., Panichella, A., Bianculli, D., & Briand, L. C. (2021, July). Log-based slicing for system-level test cases. In Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis (pp. 517-528).

[P418] Bacchelli, A., Dal Sasso, T., D'Ambros, M., & Lanza, M. (2012, June). Content classification of development emails. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 375-385). IEEE.

[P419] He, H., He, R., Gu, H., & Zhou, M. (2021, August). A large-scale empirical study on Java library migrations: prevalence, trends, and rationales. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 478-490).

[P420] Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021, May). Towards automating code review activities. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 163-174). IEEE.

[P421] Rebai, S., Amich, A., Molaei, S., Kessentini, M., & Kazman, R. (2020). Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. Automated Software Engineering, 27, 301-328.

[P422] Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., & Mockus, A. (2020, June). Detecting and characterizing bots that commit code. In Proceedings of the 17th international conference on mining software repositories (pp. 209-219).

[P423] Alizadeh, V., & Kessentini, M. (2018, September). Reducing interactive refactoring effort via clustering-based multi-objective search. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 464-474).

[P424] Basios, M., Li, L., Wu, F., Kanthan, L., & Barr, E. T. (2018, October). Darwinian data structure selection. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 118-128).

[P425] Castelein, J., Aniche, M., Soltani, M., Panichella, A., & van Deursen, A. (2018, May). Search-based test data generation for SQL queries. In Proceedings of the 40th international conference on software engineering (pp. 1220-1230).

[P426] Levin, S., & Yehudai, A. (2017, September). The co-evolution of test maintenance and code maintenance through the lens of fine-grained semantic changes. In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 35-46). IEEE.

[P427] Shi, L., Chen, C., Wang, Q., Li, S., & Boehm, B. (2017, October). Understanding feature requests by leveraging fuzzy method and linguistic analysis. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 440-450). IEEE.

[P428] Coppik, N., Schwahn, O., Winter, S., & Suri, N. (2017, October). TrEKer: Tracing error propagation in operating system kernels. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 377-387). IEEE.

[P429] Jiang, S., Armaly, A., & McMillan, C. (2017, October). Automatically generating commit messages from diffs using neural machine translation. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 135-146). IEEE.

[P430] Tomasdottir, K. F., Aniche, M., & Van Deursen, A. (2017, October). Why and how JavaScript developers use linters. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 578-589). IEEE.

[P431] Khatchadourian, R., Sawin, J., & Rountev, A. (2007, October). Automated refactoring of legacy Java software to enumerated types. In 2007 IEEE International Conference on Software Maintenance (pp. 224-233). IEEE.

[P432] Di Franco, A., Guo, H., & Rubio-Gonzalez, C. (2017, October). A comprehensive study of real-world numerical bug characteristics. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 509-519). IEEE.

[P433] Chen, B., & Jiang, Z. M. (2017, May). Characterizing and detecting anti-patterns in the logging code. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 71-81). IEEE.

[P434] Khatchadourian, R., & Masuhara, H. (2017, May). Automated refactoring of legacy Java software to default methods. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 82-93). IEEE.

[P435] Zhou, Y., & Sharma, A. (2017, August). Automated identification of security issues from commit messages and bug reports. In Proceedings of the 2017 11th joint meeting on foundations of software engineering (pp. 914-919).

[P436] Lin, Y., Sun, J., Xue, Y., Liu, Y., & Dong, J. (2017, May). Feedback-based debugging. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 393-403). IEEE.

[P437] Zhang, T., & Kim, M. (2017, May). Automated transplantation and differential testing for clones. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 665-676). IEEE.

[P438] Bohme, M., Soremekun, E. O., Chattopadhyay, S., Ugherughe, E., & Zeller, A. (2017, August). Where is the bug and how is it fixed? an experiment with practitioners. In Proceedings of the 2017 11th joint meeting on foundations of software engineering (pp. 117-128).

[P439] Palomba, F., Salza, P., Ciurumelea, A., Panichella, S., Gall, H., Ferrucci, F., & De Lucia, A. (2017, May). Recommending and localizing change requests for mobile apps based on user reviews. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 106-117). IEEE.

[P440] Dijk, R. V., Creeten, C., Ham, J. V. D., & Bos, J. V. D. (2017, August). Model-driven software engineering in practice: privacy-enhanced filtering of network traffic. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 860-865).

[P441] Soltani, M., Panichella, A., & Van Deursen, A. (2017, May). A guided genetic algorithm for automated crash reproduction. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 209-220). IEEE.

[P442] Almasi, M. M., Hemmati, H., Fraser, G., Arcuri, A., & Benefelds, J. (2017, May). An industrial evaluation of unit test generation: Finding real faults in a financial application. In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP) (pp. 263-272). IEEE.

[P443] Wu, J., Liu, S., Ji, S., Yang, M., Luo, T., Wu, Y., & Wang, Y. (2017, May). Exception beyond Exception: Crashing Android System by Trapping in" Uncaught Exception". In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP) (pp. 283-292). IEEE.

[P444] Spadini, D., Aniche, M., Bruntink, M., & Bacchelli, A. (2017, May). To mock or not to mock? An empirical study on mocking practices. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR) (pp. 402-412). IEEE.

[P445] Pascarella, L., & Bacchelli, A. (2017, May). Classifying code comments in Java open-source software systems. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR) (pp. 227-237). IEEE.

[P446] Jiang, Y., Chen, H., Qin, F., Xu, C., Ma, X., & Lu, J. (2016, November). Crash consistency validation made easy. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 133-143).

[P447] Silva, D., Tsantalis, N., & Valente, M. T. (2016, November). Why we refactor? confessions of github contributors. In Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering (pp. 858-870).

[P448] Zhang, H., Jain, A., Khandelwal, G., Kaushik, C., Ge, S., & Hu, W. (2016, November). Bing developer assistant: improving developer productivity by recommending sample code. In Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering (pp. 956-961).

[P449] Gao, F., Wang, L., & Li, X. (2016, August). Bovinspector: automatic inspection and repair of buffer overflow vulnerabilities. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 786-791).

[P450] Yoshida, H., Tokumoto, S., Prasad, M. R., Ghosh, I., & Uehara, T. (2016, July). FSX: fine-grained incremental unit test generation for C/C++ programs. In Proceedings of the 25th international symposium on software testing and analysis (pp. 106-117).

[P451] Wen, M., Wu, R., & Cheung, S. C. (2016, August). Locus: Locating bugs from software changes. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 262-273).

[P452] Rahman, M. M., Roy, C. K., Redl, J., & Collins, J. A. (2016, August). Correct: Code reviewer recommendation at github for vendasta technologies. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 792-797).

[P453] Hannebauer, C., Patalas, M., Stunkel, S., & Gruhn, V. (2016, August). Automatically recommending code reviewers based on their expertise: An empirical comparison. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 99-110).

[P454] Almhana, R., Mkaouer, W., Kessentini, M., & Ouni, A. (2016, August). Recommending relevant classes for bug reports using multi-objective search. In Proceedings of the 31st IEEE/ACM international conference on automated software engineering (pp. 286-295).

[P455] Legunsen, O., Hassan, W. U., Xu, X., Rosu, G., & Marinov, D. (2016, August). How good are the specs? A study of the bug-finding effectiveness of existing Java API specifications. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 602-613).

[P456] Wang, S., Liu, T., & Tan, L. (2016, May). Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering (pp. 297-308).

[P457] Xiao, L., Cai, Y., Kazman, R., Mo, R., & Feng, Q. (2016, May). Identifying and quantifying architectural debt. In Proceedings of the 38th international conference on software engineering (pp. 488-498).

[P458] Yu, T., Qu, X., & Cohen, M. B. (2016, May). VDTest: An automated framework to support testing for virtual devices. In Proceedings of the 38th International Conference on Software Engineering (pp. 583-594).

[P459] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016, May). Automated parameter optimization of classification techniques for defect prediction models. In Proceedings of the 38th international conference on software engineering (pp. 321-332).

[P460] Martinez, J., Ziadi, T., Bissyande, T. F., Klein, J., & Le Traon, Y. (2015, November). Automating the extraction of model-based software product lines from model variants (T). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 396-406). IEEE.

[P461] Narasimhan, K., & Reichenbach, C. (2015, November). Copy and paste redeemed (t). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 630-640). IEEE.

[P462] Choudhary, S. R., Gorla, A., & Orso, A. (2015, November). Automated test input generation for android: Are we there yet?(e). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 429-440). IEEE.

[P463] Vargas-Baldrich, S., Linares-Vasquez, M., & Poshyvanyk, D. (2015, November). Automated tagging of software projects using bytecode and dependencies (n). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 289-294). IEEE.

[P464] Person, S., & Elbaum, S. (2015, November). Test analysis: Searching for faults in tests (n). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 149-154). IEEE.

[P465] Kevic, K., Walters, B. M., Shaffer, T. R., Sharif, B., Shepherd, D. C., & Fritz, T. (2015, August). Tracing software developers' eyes and interactions for change tasks. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 202-213).

[P466] Foo, K. C., Jiang, Z. M., Adams, B., Hassan, A. E., Zou, Y., & Flora, P. (2015, May). An industrial case study on the automated detection of performance regressions in heterogeneous environments. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 159-168). IEEE.

[P467] Goldstein, M., & Segall, I. (2015, May). Automatic and continuous software architecture validation. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 59-68). IEEE.

[P468] Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M. R., & Zhang, D. (2015, May). Learning to log: Helping developers make informed logging decisions. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 415-425). IEEE.

[P469] Dhar, A., Purandare, R., Dhawan, M., & Rangaswamy, S. (2015, August). CLOTHO: Saving programs from malformed strings and incorrect string-handling. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 555-566).

[P470] Casalnuovo, C., Devanbu, P., Oliveira, A., Filkov, V., & Ray, B. (2015, May). Assert use in github projects. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 755-766). IEEE.

[P471] Ye, X., Bunescu, R., & Liu, C. (2014, November). Learning to rank relevant files for bug reports using domain knowledge. In Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering (pp. 689-699).

[P472] Allamanis, M., & Sutton, C. (2014, November). Mining idioms from source code. In Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering (pp. 472-483).

[P473] Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014, November). Are mutants a valid substitute for real faults in software testing?. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 654-665).

[P474] Jing, X. Y., Ying, S., Zhang, Z. W., Wu, S. S., & Liu, J. (2014, May). Dictionary learning based software defect prediction. In Proceedings of the 36th international conference on software engineering (pp. 414-423).

[P475] Schiller, T. W., Donohue, K., Coward, F., & Ernst, M. D. (2014, May). Case studies and tools for contract specifications. In Proceedings of the 36th International Conference on Software Engineering (pp. 596-607).

[P476] Boci*fá*, I., & Bultan, T. (2014, May). Inductive verification of data model invariants for web applications. In Proceedings of the 36th International Conference on Software Engineering (pp. 620-631).

[P477] Le, W., & Pattison, S. D. (2014, May). Patch verification via multiversion interprocedural control flow graphs. In Proceedings of the 36th International Conference on Software Engineering (pp. 1047-1058).

[P478] Lin, Y., Xing, Z., Xue, Y., Liu, Y., Peng, X., Sun, J., & Zhao, W. (2014, May). Detecting differences across multiple instances of code clones. In Proceedings of the 36th International Conference on Software Engineering (pp. 164-174).

[P479] Nadi, S., Berger, T., Kastner, C., & Czarnecki, K. (2014, May). Mining configuration constraints: Static analyses and empirical results. In Proceedings of the 36th international conference on software engineering (pp. 140-151).

[P480] Campos, J., Arcuri, A., Fraser, G., & Abreu, R. (2014, September). Continuous test generation: Enhancing continuous integration with automated test generation. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering (pp. 55-66).

[P481] Wu, Y., Sun, J., Liu, Y., & Dong, J. S. (2013, November). Automatically partition software into least privilege components using dynamic data dependency analysis. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 323-333). IEEE.

[P482] Coker, Z., & Hafiz, M. (2013, May). Program transformations to fix C integers. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 792-801). IEEE.

[P483] Garg, P., Ivan*fçifá*, F., Balakrishnan, G., Maeda, N., & Gupta, A. (2013, May). Feedback-directed unit test generation for C/C++ using concolic execution. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 132-141). IEEE.

[P484] Qiu, D., Li, B., & Su, Z. (2013, August). An empirical analysis of the co-evolution of schema and code in database applications. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (pp. 125-135).

[P485] Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: a multi-objective approach. Automated Software Engineering, 20, 47-79.

[P486] Xiao, X., Paradkar, A., Thummalapenta, S., & Xie, T. (2012, November). Automated extraction of security policies from natural-language software documents. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (pp. 1-11).

[P487] Zhou, J., Zhang, H., & Lo, D. (2012, June). Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In 2012 34th International conference on software engineering (ICSE) (pp. 14-24). IEEE.

[P488] Mesbah, A., & Mirshokraie, S. (2012, June). Automated analysis of CSS rules to support style maintenance. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 408-418). IEEE.

[P489] Grechanik, M., Fu, C., & Xie, Q. (2012, June). Automatically finding performance problems with feedback-directed learning software testing. In 2012 34th international conference on software engineering (ICSE) (pp. 156-166). IEEE.

[P490] Xiong, Y., Hubaux, A., She, S., & Czarnecki, K. (2012, June). Generating range fixes for software configuration. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 58-68). IEEE.

[P491] Servant, F., & Jones, J. A. (2012, June). Whosefault: automatic developer-to-fault assignment through fault localization. In 2012 34th International conference on software engineering (ICSE) (pp. 36-46). IEEE.precise identification of problems for structural test generation

[P492] Fraser, G., & Arcuri, A. (2012, June). Sound empirical evidence in software testing. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 178-188). IEEE.

[P493] Xiao, X., Xie, T., Tillmann, N., & De Halleux, J. (2011, May). Precise identification of problems for structural test generation. In Proceedings of the 33rd International Conference on Software Engineering (pp. 611-620).

[P494] Dig, D., Marrero, J., & Ernst, M. D. (2011, May). How do programs become more concurrent: a story of program transformations. In Proceedings of the 4th International Workshop on Multicore Software Engineering (pp. 43-50).

[P495] Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011, September). Relink: recovering links between bugs and changes. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (pp. 15-25).

[P496] Ivan*fçifá*, F., Balakrishnan, G., Gupta, A., Sankaranarayanan, S., Maeda, N., Tokuoka, H., ... & Miyazaki, Y. (2011, November). DC2: A framework for scalable, scope-bounded software verification. In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011) (pp. 133-142). IEEE.

[P497] Dulz, W. (2011, May). A comfortable testplayer for analyzing statistical usage testing strategies. In Proceedings of the 6th International Workshop on Automation of Software Test (pp. 36-42).

[P498] Zhong, H., Zhang, L., Xie, T., & Mei, H. (2011). Inferring specifications for resources from natural language API documentation. Automated Software Engineering, 18, 227-261.

[P499] Pham, N. H., Nguyen, T. T., Nguyen, H. A., & Nguyen, T. N. (2010, September). Detection of recurring software vulnerabilities. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 447-456).

[P500] Di Penta, M., German, D. M., & Antoniol, G. (2010, May). Identifying licensing of jar archives using a code-search approach. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010) (pp. 151-160). IEEE.

[P501] Austin, A., Smith, B., & Williams, L. (2010, May). Towards improved security criteria for certification of electronic health record systems. In Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care (pp. 68-73).

[P502] Kanade, A., Alur, R., Rajamani, S., & Ramanlingam, G. (2010, November). Representation dependence testing using program inversion. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (pp. 277-286).

[P503] Buse, R. P., & Weimer, W. R. (2010, September). Automatically documenting program changes. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 33-42).

[P504] German, D. M., Manabe, Y., & Inoue, K. (2010, September). A sentence-matching method for automatic license identification of source code files. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 437-446).

[P505] Ossher, J., Bajracharya, S., & Lopes, C. (2010, May). Automated dependency resolution for open source software. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010) (pp. 130-140). IEEE.

[P506] Marin, M., van Deursen, A., Moonen, L., & van der Rijst, R. (2009). An integrated crosscutting concern migration strategy and its semi-automated application to JHotDraw. Automated Software Engineering, 16, 323-356.

[P507] Antkiewicz, M., Tonelli Bartolomei, T., & Czarnecki, K. (2009). Fast extraction of high-quality framework-specific models from application code. Automated Software Engineering, 16, 101-144.

[P508] Wasylkowski, A., Zeller, A., & Lindig, C. (2007, September). Detecting object usage anomalies. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 35-44).

[P509] Grechanik, M., McKinley, K. S., & Perry, D. E. (2007, September). Recovering and using use-case-diagram-to-source-code traceability links. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 95-104).

[P510] Ramanathan, M. K., Grama, A., & Jagannathan, S. (2006, September). Sieve: A tool for automatically detecting variations across program versions. In 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06) (pp. 241-252). IEEE.

[P511] Kim, S., Pan, K., & Whitehead Jr, E. J. (2006, May). Micro pattern evolution. In Proceedings of the 2006 international workshop on Mining software repositories (pp. 40-46).

[P512] Neamtiu, I., Hicks, M., Stoyle, G., & Oriol, M. (2006). Practical dynamic software updating for C. ACM SIGPLAN Notices, 41(6), 72-83.

[P513] Hill, R., & Rideout, J. (2004, September). Automatic method completion. In Proceedings. 19th International Conference on Automated Software Engineering, 2004. (pp. 228-235). IEEE.

[P514] Xie, Y., Chou, A., & Engler, D. (2003, September). Archer: using symbolic, path-sensitive analysis to detect memory access errors. In Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 327-336).

[P515] Turnu, I., Melis, M., Cau, A., Marchesi, M., & Setzu, A. (2004, November). Introducing tdd on a free libre open source software project: a simulation experiment. In Proceedings of the 2004 workshop on Quantitative techniques for software agile process (pp. 59-65).