Université de Mons
Faculté des Sciences
Département d'Informatique

Identifying development bots in social coding platforms

Mehdi Golzadeh

A dissertation submitted in fulfilment of the requirements of the degree of *Docteur en Sciences*

Advisors

Prof. Dr. Tom Mens	University of Mons, Belgium

Jury

Prof. Dr. Eleni Constantinou	University of Cyprus (UCY), Cyprus
Dr. Alexandre Decan	University of Mons, Belgium
Prof. Dr. Bruno Quoitin	University of Mons, Belgium
Prof. Dr. Mairieli Wessel	Radboud University, The Netherlands

June 22, 2023

Acknowledgements

As I bring this significant chapter of my life to a close, I am grateful to all those who have supported me through my four-year journey. Your guidance, encouragement, and assistance have been invaluable, and I cannot express my appreciation enough.

First and foremost, I would like to express my heartfelt gratitude to Tom, my advisor, for his unwavering support and guidance throughout my PhD journey. Despite the challenges posed by the COVID-19 pandemic, he provided me with invaluable feedback and support in every aspect of my work. Working under his supervision has been an incredible experience, and I feel privileged to have had the opportunity to work with such a dedicated professor who motivated me to exceed my own expectations.

I would also like to extend my thanks to Alex, whose assistance and feedback were crucial in helping me to refine my research and improve the quality of my work. His knowledge and expertise in the field of software engineering were invaluable, and I am deeply grateful for his contributions. Additionally, I would like to express my appreciation to all of my colleagues and co-authors of my papers, whose collaboration and insights have greatly enriched my research and understanding of the field.

To the members of the jury, I express my sincere gratitude for your willingness to review my dissertation and provide constructive feedback. Your insightful comments and valuable suggestions have significantly improved the quality of my work. I would like to acknowledge Eleni for our collaboration on papers and her invaluable advice. Maireli, thank you for sharing your knowledge and organizing exceptional workshops on bots in software engineering. Your papers have been a great source of inspiration for me. Lastly, a special thanks to Bruno for providing an external perspective on my work and engaging in technical discussions.

I am deeply grateful to my wife for her unwavering love and support throughout my PhD journey. Her encouragement, patience, and positive attitude have been a constant source of inspiration and motivation. I feel truly blessed to have such a supportive partner who has always been there for me and my work.

I would like to express my immense gratitude to my parents, who have been a constant source of support and encouragement throughout my entire life, including these past four years of my academic journey. Their love, encouragement, and guidance have been invaluable, and I would not have been able to achieve what I have today without them. In addition, I am thankful to my brother, sister, and their families for being a constant source of love, support, and laughter, and for sharing so many wonderful moments with me.

Abstract

Social coding platforms such as GitHub and mechanisms such as pull requests, code reviews, issue reports, and commenting allow developers to contribute to several different software projects without any geographical restrictions. Developers use software bots in order to reduce their workload by automating tasks such as reporting continuous integration failures and test coverage, checking license agreement signing, triaging issues, reviewing code and pull requests, updating dependencies, etc. As such, bots have become very common in software development practices and they are widely used in software projects. While bots offer many benefits, they also have limitations and potential drawbacks, making it necessary to study their impact on software development. However, there is no automatic way to distinguish human accounts from bot accounts in software development repositories. Studies that investigated the effect of bots relied on a manual inspection or simple heuristics like the presence of “bot” in the account name to identify bots in software repositories, but these methods are either inefficient or produce many false negatives.

In this dissertation the main goal is to fill this gap by conducting empirical studies about software development bots and to develope tools and techniques to automatically identify them. We shed light on the fact that bots are prevalent in software repositories and commonly active contributors to such projects. We developed a ground truth dataset of GitHub bot accounts and human accounts, characterized bots based on their pull request and issue commenting activities, and trained a classification model to identify bots based on the discovered characteristics. Since the model was able to identify most bots effectively, we included it as an integrated part of an open-source tool called BoDeGHa to make it easier for practitioners and researchers to use.

We extended the classification model to be able to identify bots in commit data. We also developed another tool that predicts the type of Git accounts based on their commit message. Additionally, we created a classification model using natural language processing techniques to identify bot activities. We further developed a classification model to improve the predictions of BoDeGHa. Finally, we carried out a comparison between existing bot detection methods and provided an ensemble model that combines all these techniques to identify bots. In conclusion, this dissertation provides valuable insights into the prevalence and impact of bots in software development and offers practical tools and techniques to automatically identify them, which can benefit both practitioners and researchers in the field.

Contents

1 Introduction **1**
1.1 Thesis objectives 2
1.2 Background 3
1.3 Version control systems 4
1.4 Collaborative software development 5
1.5 Pull-based development 6
1.6 Development workflow automation 7
1.6.1 Continuous integration 7
1.6.2 Continuous deployment and delivery 7
1.6.3 Development bots 8
1.7 Thesis statement 9
1.8 Contributions 9
1.9 Thesis structure 10
1.10 Additional publications 13

2 State of the art **17**
2.1 Continuous integration systems 18
2.2 Bots: evolution and origin 21
2.3 Bots: Definitions and classification schemes 23
2.4 Software development bots 25
2.4.1 Bots in software engineering 25
2.4.2 Bots in GitHub 26
2.4.3 On the need for bot identification techniques 27
2.5 Bot identification 28

3 Workflow automation tools **31**
3.1 On the prevalence of Continuous integration (CI) in GitHub . . 32

3.1.1 Data extraction 33
3.1.2 How did the CI landscape evolve? 35
3.1.3 What are the most frequent combinations of CIs? 39
3.1.4 How frequently are CIs being replaced by an alternative? 42
3.1.5 How has the CI landscape changed since GitHub Actions was introduced? 43
3.2 On the prevalence of bots in GitHub 47
3.2.1 Acknowledging contributions in collaborative development 48
3.2.2 Distinguishing bots from humans 50
3.2.3 Evidence of bot contributions i n popular software projects 50
3.3 Summary and conclusions 52

4 Distinguishing characteristics of bots 55
4.1 Terminology 56
4.2 Data extraction 56
4.3 Data labelling and rating process 58
4.4 Characteristics of bots 62
4.4.1 Text distance between comments 62
4.4.2 Repetitive comment patterns 65
4.4.3 Inequality between comments in patterns 66
4.4.4 Number of comments 68
4.4.5 Number of empty comments 68
4.5 Summary and conclusions 69

5 A classification model for bot detection 71
5.1 A classification model to identify bot accounts 72
5.2 Evaluation of the model 75
5.3 The BoDeGHa bot detection tool 77
5.4 Limitations of the bot detection model 80
5.5 Threats to validity 83
5.6 Summary and conclusions 85

6 Improving and extending bot detection techniques 87
6.1 A classification model based on commit messages 88
6.1.1 Initial classification model 89
6.1.2 Data extraction 90
6.1.3 Model and evaluation 91
6.1.4 The BoDeGiC bot detection tool 93
6.1.5 Discussion 94
6.2 A classification model to identify bot activities 96

6.2.1 Data extraction 96
6.2.2 Model construction 97
6.2.3 Model evaluation 99
6.2.4 Discussion 101
6.3 A classification model based on multiple repositories 102
6.3.1 Data extraction 104
6.3.2 How frequently are contributors active in multiple repositories? 104
6.3.3 How frequently do contributors have diverging or incomplete predictions? 105
6.3.4 To which extent can we fix diverging predictions? 106
6.3.5 To which extent can we complete predictions? 108
6.4 Accuracy of bot detection techniques 109
6.4.1 Data extraction 111
6.4.2 How accurate are bot detection techniques? 112
6.4.3 How prevalent are bots among active contributors? . . . 113
6.4.4 How active are bots in terms of commits? 115
6.5 Summary and conclusions 115

7 Conclusions 119
7.1 Contributions 120
7.1.1 Created datasets 120
7.1.2 Classification models to identify bot accounts 121
7.1.3 Bot detection tools 122
7.2 The Impact of the Contributions 123
7.3 Future research perspectives 124
7.3.1 Bot identificaton and dataset improvement 124
7.3.2 Enhance software development processes 127
7.3.3 Bot behavior optimization 128

CHAPTER 1
Introduction

"What you seek is seeking you."

Molana Rumi

I started my PhD studies in 2019 after a decade of practical software development experience. Throughout my career as software developer and in my previous academic studies I have learnt and used a wide variety of different software development methodologies, technologies, design patterns, and tools. My experience extends to collaborative software development with other developers using advanced version control tools and social coding platforms. I have been using GitHub as a social coding platform and a codebase for collaborating with other software developers.

Today, GitHub has become the most popular platform to host open-source software projects. It is also an excellent source of data to empirically study contemporary software development practices and projects. GitHub data is therefore being used by many researchers to extract knowledge to analyse, understand and improve software development processes.

With my software development background and data analytics experience I decided to pursue doctoral studies in this important field of software development analytics. To pursue my PhD goal I started to empirically analyse development automation tools. This dissertation presents my research that I have conducted over the last four years. The dissertation explains my research objectives, thesis statement, methodology, findings and software artifacts as well as as discussions around the subject.

1.1. Thesis objectives

The emergence of social coding platforms such as GitHub has revolutionized collaborative software development practices in the last decade, especially in the open-source community. Developers collaborate in many different software projects without geographical restrictions. They contribute code using mechanisms such as pull requests, perform code reviews, submit issues and communicate with other developers through commenting mechanisms in social coding platforms. This change in style of development has significantly increased the workload of software developers and has made software maintenance more challenging. In order to reduce this workload, developers tend to adopt workflow automation tools such as bots. Such tools facilitate collaborative software development by performing repetitive tasks such as running tests, building, deploying and releasing new versions, reporting code coverage, performing code quality checks and welcoming new contributors.

As such, workflow automation and bots have become a key aspect of contemporary collaborative software development practices. Thus, it is important to study the evolution and impact of the presence of such tools on the collaborative development process. There is, however, no automatic method available for identifying automated bot activities in software development repositories. Research in this area focuses primarily on proposing bots, as well as analysing specific bots and their impact. Studies that investigated the effect of bots in software engineering process, relied on a manual method to identify bots in software repositories and there is no automatic method to accurately identify bots. This dissertation aims to address the lack of the presence of an automatic bot identification method by proposing classification models and tools to detect them.

In this regard, we have conducted several empirical studies in order to reveal the prevalence of automated development activities. First, we analyze the evolution of continuous integration systems as the backbone of such automatic tools and track the historical changes in their trends of usage. Then, we show that bots are among the most active contributors and existing methods are not able to effectively identify them. Next, we create a large ground-truth dataset of bot and human accounts based on their activities in GitHub pull request and issue comments. We characterized automated activities and we identified four features that allowed us to create a classification model and evaluate it on this ground-truth dataset. We also developed an open-source command-line tool to detect bots in GitHub repositories. Finally, we report about the studies to improve the existing models, extending the model to identify bots in commit activities and the associated tool, and providing more advanced classification models to detect bots and human activities.

1.2. Background

Software development includes several phases and iterations, including collecting requirements, analysis, design, creating applications, testing them, deploying, and maintaining the released product. These steps are collectively called the *Software Development LifeCycle (SDLC)*. A piece of software goes through all these processes from concept to launch and continues to evolve after the initial release. Each step in this process typically produces an output (e.g., an idea, a document, a diagram, or a working piece of software), which functions as an input for the next step (Ince & Andrews, 1998).

Despite their sequential nature, these steps do not end once the software has been delivered to stakeholders, and the software development team repeats them many times to enhance the product (Davis et al., 1988). A full software development cycle is followed during each iteration which minimizes the overall risk of failure and allows the project to adapt to changes quickly. An SDLC model aims to ensure successful product delivery through a series of steps. Some of the prominent SDLC models are the waterfall model, iterative model, spiral model, etc (Davis et al., 1988; Sultanía, 2015).

Development of software is a collaborative problem-solving activity that requires knowledge acquisition, information sharing, and integration, as well as minimal communication breakdowns (Bird, 2011; Tymchuk et al., 2014). Developing software systems successfully requires collaboration across every phase of the SDLC. Developers typically perform many different tasks during a software development process, including creating software artifacts (source code, models, documentation, and test scenarios), managing and coordinating their work, as well as communicating with other developers (Whitehead, 2007). Meanwhile, the software artifacts go through changes several times by different developers in different steps. Hence, an orchestrator tool is made for facilitating the collaboration between developers. A *Version Control System (VCS)* (Tichy, 1985) is a tool that facilitates the collaboration in a software project by tracking changes made to the source code and allowing developers to work on the codebase concurrently.

1.3. Version control systems

Change is a crucial part of software development as a software product encounters many modifications over the course of its lifetime. In order to track these changes version control systems (VCS) are commonly used. The main purpose of VCS is collaboration (Koc & Tansel, 2011). Early VCSs were developed in the 1970s to facilitate the continuous modification of source code by multiple developers (Rochkind, 1975). A VCS enables developers to collaborate on a shared codebase without interrupting each other. A VCS provides the ability to track the evolution of files, and the state of code at a specific time and restore code to a specific time if required. A VCS is not just about code changes but it also stores, who, when, and (possibly) why this change has happened (Koc & Tansel, 2011).

VCSs come in two forms, Centralized Version Control System (CVCS) and Distributed Version Control System (DVCS) (Otte, 2009). In the centralised model users work with a central repository while in the distributed model there is no central repository but each user works on their local repository. Concurrent Versions System (CVS) and Subversion have been very popular CVCS tools in the past, next to commercial tools such as Perforce, ClearCase and many more (de Alwis & Sillito, 2009; Collin-Sussmen et al., 2002). Git, Mercurial, Bazaar, and BitKeeper are well-known examples of DVCS (Collin-Sussmen et al., 2002). CVCSs come with some disadvantages. Given that the code base needs to be stored on a central server, there is a risk of losing all changes in the event of server corruption, unsolicited changes could disrupt the development process, and if the server goes down, developers will not be able to save their versioned changes. Most open-source software projects have therefore adopted distributed version control systems to record, manage, and propagate source code changes to avoid problems caused by the aforementioned risks (Somasundaram, 2013).

In 2005, Linus Torvalds released a new version control system called Git. Since Git is the most popular DVCS and used by most software development platforms, we focus in this section specifically on it. Torvalds' goal was to develop a version control tool with an emphasis on speed and support for distributed and non-linear way of coding which were not supported by existing CVCSs. Although Git was initially not intended as a general-purpose tool, it provided generic concepts that enabled agile development processes (Just et al., 2016). This has made Git one of the most widely used DVCS (Otte, 2009) nowadays, both in the open-source community and the industry. It allows developers to work offline in their own branch more efficiently by providing a large set of powerful features such as parallel branches and provides opportunities for teams to increase agility.

In addition to the general features that all VCSs provide, Git offers several additional features that make it an attractive DVCS. It allows the development team to manage parallel development activities with branches in order to isolate individual work and merge the result back to the main branch after finishing the tasks. This feature especially makes it more interesting than its competitor Mercurial. Git also provides more level of granularity when merging operations which means it allows developers to more easily manage and resolve conflicts that may arise when merging changes from multiple sources (Just et al., 2016).

1.4. Collaborative software development

The demand for better software products, faster services, and the growth in the number of companies providing these solutions have led in the past decades to companies distributing their development processes overseas, also known as global software development. It is common for a software project to have developers in different locations, often on more than one continent (Herbsleb, 2007). Software development is considered by researchers to be a complicated activity, since it involves many risks and uncertainties. Adding the physical and temporal distance between the participants of the development process to the traditional difficulty of developing software increases the challenges of the process and creates new difficulties (Herbsleb, 2007).

To ensure the success of a software development project, collaboration between developers is necessary at all stages of the development cycle. It has been a challenge in global software engineering research and practice to understand developer collaboration practices as well as communication and messaging tools used to facilitate developers working together in software development tasks (Dabbish et al., 2012; Gousios et al., 2016, 2015; Lanubile et al., 2010; Constantino et al., 2020). Researchers proposed and evaluated tools (Bjorn et al., 2014; Lanubile et al., 2010) and models (Arciniegas-Mendez et al., 2017; Steinmacher et al., 2019), in order to facilitate collaboration and help developers with collaborative tasks.

Open-source software (OSS) development is a notable model of collaborative development (Laurent, 2004; Riehle et al., 2009) that is increasingly relying on third-party contributors. By providing workflows that allow developers to work in isolation, as well as to collaborate by integrating their work with other developers, DVCSs have enabled the open-source movement to generate more contributions (Kalliamvakou et al., 2014a). Using branching and merging, developers can separate their work by purpose in a dedicated workspace and merge their code frequently. While DVCSs allow developers to work in isolation and merge their changes into a software project, they still require a central code repository where the code is placed. Code hosting platforms (such as SourceForge) or social coding platforms (such as GitHub, GitLab and BitBucket) provide such a central space.

Collaboration extends software development from only a technical activity to a social phenomenon (Tsay et al., 2014a). Social activities such as commenting and reviewing play an essential role in such an environment and are sometimes as critical as technical activities (Tsay et al., 2014b). Social coding platforms combine social and technical requirements of collaboration in software development in a single environment (Metz, 2015). They suggest an open workflow where collaborators can participate in projects in a variety of ways. For example, contributors can contribute to discussions regarding bugs and features through an issue tracking system, or make changes to a project directly through Git, and indirectly through the pull request mechanism and let it be reviewed and accepted by maintainers through a code review mechanism (Gousios et al., 2014; Tsay et al., 2014a,b). Among the existing social coding platforms, GitHub is the most popular one (Varuna & Mohan, 2019).

1.5. Pull-based development

Git and GitHub have revolutionized software development by facilitating distributed collaborative software development (Arora et al., 2016). GitHub is by far the largest social coding platform, hosting the development history of millions of collaborative software repositories, and accommodating over 73 million users in 2021 (GitHub, 2021). Features provided by GitHub make it a lot easier for individuals and teams to use Git as a DVCS and collaborate in software development through a web interface. The reason for GitHub's reputation as a social coding platform is that, in addition to providing a Git repository host and all of the Git capabilities, it also allows developers to contribute code through a Pull Request (PR).

The novelty of this approach lies in the fact that the development effort is decoupled from the decision to incorporate the results of the development into the codebase (Gousios et al., 2014, 2015, 2016). In the pull-based development model, one can distinguish between direct and indirect developer contributions to a project. Direct contributions come from a typically small group of developers with direct access to the main repository, while indirect contributions come from developers who fork the main repository, update their copies locally, and submit pull requests. The indirect contribution method allows contributors to develop and submit their code without having to deal with repository maintenance concerns and enables maintainers to review the code, test it, request changes, and finally integrate it into the codebase without getting involved in code development.

1.6. Development workflow automation

1.6.1 Continuous integration

It's been more than a decade that software developers work on the same project without geographical restrictions and commit codes to the code base regularly in order to check the integration issues and to ensure that the latest version of the code is available for other developers (Jiménez et al., 2009). *CI* is a software development practice in which a tool continuously integrates, tests, and builds developer code after each commit or at specific times, and reports the build's results (Vasilescu et al., 2014). CI systems reduce the integration failures by applying quality control checks to the code contributions. Duvall et al. (Duvall et al., 2007) reported that this results in a faster development process and improved software quality.

CI systems come in two forms. They are either offered as cloud-based services that enable developers to use the continuous integration process only by connecting their repository to the service, or they are on-premise tools that require a server and setup. Many CI systems exist, some of the most well-known ones are Travis, CircleCI, Jenkins, GitLab CI and GitHub Actions.

GitHub's implementation of the pull-based development model makes it possible for anyone to contribute code to any repository within a minute. This provides exceptionally low entry barriers for potential contributors but also presents testing-related challenges reported by GitHub project owners (Pham et al., 2013). These challenges led software developers to make use of CI systems to facilitate automated testing. Whenever a commit or pull request is received, the contribution is automatically merged into a testing branch, the existing test suite is run, and the author and owner of the project are notified.

1.6.2 Continuous deployment and delivery

Continuous delivery and deployment are continuous practices that enable organizations to frequently and reliably release new features (Shahin et al., 2017).

The terms continuous delivery and continuous deployment are sometimes used interchangeably in academic and industrial contexts, but each has its own definition (Fitzgerald & Stol, 2017).

Continuous DElivery (CDE) ensures that an application is always in a production-ready state following the successful passing of automated tests and quality checks (Weber et al., 2016). Chen et al. (Chen, 2015) reported that, CDE follows practices that offer benefits such as reducing deployment risk, lowering costs to deliver software packages, and getting faster user feedback.

Continuous Deployment (CD) automates the deployment of an application to production or to a customer's environment on a continuous basis (Weber et al., 2016). An important distinction between CD and CDE is the production environment. Skelton et al. (Skelton & O'Dell, 2016) consider CD as a push-based approach and there is no manual step before the production, while CDE approach is a pull-based approach for which a business decides what and when to deploy. Although CDE practices can be applied to all types of systems and organizations, CD practices may only be relevant for certain types of organizations or systems.

Hereafter when we use the term *continuous integration*, or *CI* for short, we refer to tools and services dedicated to all activities of CI and CD/CDE, either cloud-based or on-premise.

1.6.3 Development bots

As mentioned before, social coding platforms such as GitHub have taken the collaborative nature of open-source software development to a new level, by integrating mechanisms such as issue reporting, pull requests, commenting and reviewing support into distributed version control tools (Gousios et al., 2014; Tsay et al., 2014a,b). According to (Gousios et al., 2016) the pull-based development method has significantly increased the workload of repository maintainers to communicate with other contributors, review source code, deal with contributor license agreement issues, explain project guidelines, run tests and build code, and merge pull requests.

To reduce this workload, repository maintainers have been adopting automated tools to perform repetitive tasks in the development process (Wessel et al., 2018b), such as updating dependencies (Mirhosseini & Parnin, 2017) (e.g., *dependabot*) and fixing vulnerabilities (e.g., *snykbot*), improving code reviews (e.g., *Review bot*) (Balachandran, 2013) and documenting code refactorings (Rebai et al., 2019). Such tools are commonly known as *development bots* (Erlenhov et al., 2020b), or *bots* for short. They are generally seen as a promising approach to deal with the ever-increasing complexity of contemporary distributed software development. Bots are also sometimes a representation of the presence of a CI system in a project, for example, *CodeCov* is a code coverage analyser that reports the result of running test suites in CI (Zhai et al., 2019). Bots have been reported to automate a significant amount of development workload, and many of them are among the most active project contributors (Golzadeh et al., 2022a). However, many of these bot accounts are not marked as bot in GitHub since they use regular accounts like other developers. This can impose issues for both researchers and practitioners.

1.7. Thesis statement

In the light of all the above explanations and the increasing use of bots in the GitHub social coding platform, there is a need to understand their activities and provide automatic methods to identify bots. Appropriate bot identification techniques and tools can help practitioners and researchers to easily identify such bots and avoid overlooking the implications of their use, both in practice and while studying software engineering data.

When I started my PhD research, there were no available automatic techniques to accurately identify GitHub bots. Bot identification was achieved only through a manual investigation or very simple heuristics such as the presence of the string "bot" in the account name. Such heuristics are very shallow and lead to a lot of incorrect predictions. It was therefore necessary to propose better techniques for identifying bots in social coding platforms like GitHub. This has led me to formulate the following thesis statement:

Thesis statement

Development bots are increasingly used to automate more and more aspects of collaborative open-source software development. This raises the need to accurately identify bots and their automated activities in social coding platforms.

1.8. Contributions

This dissertation contributes to the body of software engineering knowledge by providing insights about CIs and bot activities, datasets, classification models and corresponding tools. Following is an overview of all contributions:

- A dataset of CI usage in the development repository of 91,810 npm packages in GitHub.
- Insights regarding the usage and evolution of CI tools in GitHub repositories, including CI churn, migration between CIs, co-usage of CIs, and the effect of the introduction of GitHub Actions (GitHub's integrated CI system) on the usage of other CIs.
- Insights regarding the usage of bots in GitHub repositories and evidence that many bots are active contributors, which may pose different research and practice challenges.
- A ground-truth dataset of bot accounts in GitHub.
- Characterisation of bot activities in GitHub pull request and issue comments in order to distinguish bot activity from human activity.
- A classification model to detect bot based on their commenting activity in GitHub issue and pull request comments and the corresponding command-line tool called BoDeGHa.
- A classification model to detect bots that are active in commits, based on their commit messages and the corresponding command-line tool called BoDeGiC.
- A probabilistic classification model to distinguish bot activity from human activity in GitHub. This classification model makes it possible to identify accounts that have been shared between a human and a bot known as mixed accounts.
- A classification model to improve predictions of BoDeGHa by using diverging predictions of accounts that are active in multiple repositories.
- A comparison of five available bot identification techniques and ensemble model that uses that five different bot identification in a single classifier in order to improve bot identification.

1.9. Thesis structure

This dissertation is organized into 7 chapters. The current chapter discussed the basics and thesis objectives and introduces the structure of the thesis as well as the published articles that I have co-authored. As a result of my research the reader will find a literature review of the state of the art in Chapter 2. Detailed results and findings of my studies are presented in the Chapters 3, 4, 5 and 6. Chapter 7 summarizes the main contributions of this dissertation and discusses future work. More specifically:

Chapter 2 presents and discusses the state of the art in those research domains that are closely related to my work. This comprises an introduction to collaborative software development and social coding platforms at the beginning of the chapter. The chapter continues by discussing about studies related to continuous integration tools and services. After that, an overview of the state of the art on bots in software engineering is presented. The final section introduces studies related to the implications of the use of such automation tools.

Chapter 3 presents the results of an empirical analysis of the CI landscape on GitHub. It presents evidence of how the GitHub CI landscape has evolved and how GitHub projects use CIs simultaneously or migrate to other CIs. The chapter also narrows down on how the introduction of GitHub Actions has changed the GitHub CI landscape. In addition, we discuss contributor acknowledgement in GitHub and the fact that bots are among the active contributors to GitHub projects, so identifying and understanding them is crucial for both practitioners and researchers. The results presented in this chapter are based on the following published peer-reviewed scientific articles:

- Golzadeh, M., Decan, A., & Mens, T. (2022) On the rise and fall of CI services in GitHub. In International Conference on Software Analysis, Evolution, and Reengineering (SANER),
  DOI 10.1109/SANER53432.2022.00084

- Golzadeh, M., Mens, T., Decan, A., Constantinou, E., & Chidambaram, N.(2022) Recognizing bot activity in collaborative software development. IEEE Software special issue on software bots,
  DOI 10.1109/MS.2022.3178601

Chapter 4 describes a step-by-step process to create a ground-truth dataset of bot and human accounts in a large set of GitHub repositories. While creating this ground-truth we observed some specific behavior in bot accounts. For example, we noticed that bots tend to use repetitive messages and they rarely submit empty comments due to the nature of their tasks. This chapter discusses these characteristics in detail and describes how they can be used to develop an automated technique to distinguish bots from human contributors. The results presented in this chapter are based on the following published peer-reviewed scientific articles and dataset:

- Golzadeh, M., Legay, D., Decan, A., & Mens, T. (2020) Bot or not? Detecting bots in GitHub pull request activity based on comment similarity. International ICSE Workshop on Bots in Software Engineering (BotSE), DOI 10.1145/3387940.3391503

- 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, DOI 10.1016/j.jss.2021.110911

- Golzadeh, M., Decan, A., Legay, D., & Mens, T. (2021) A ground-truth dataset to identify bots in GitHub, DOI 10.5281/zenodo.4000388

Chapter 5 explains how we trained a classification model for identifying bots using the features presented in Chapter 4, along with the evaluation of the model. Based on this model we developed and released an open-source command-line tool called BoDeGHa that allows practitioners and researchers to use the classification model in practice. Lastly, we discuss the limitations of our study as well as threats to their validity. The results presented in this chapter are based on the following published peer-reviewed scientific article and an associated tool that we developed:

- 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, DOI 10.1016/j.jss.2021.110911

- BoDeGHa: An automated tool to identify bots in GitHub repositories by analysing pull request and issue comments. https://github.com/mehdigolzadeh/BoDeGHa

Chapter 6 presents our effort to overcome the limitations of the model presented in Chapter 5 in order to improve bot identification. We achieved this goal both by improving the model of Chapter 5 and by developing new classification models. We extended our approach to work with other kind of software development activities. We also implemented a more fine-grained model to classify individual bot activities and human activities instead of classifying GitHub accounts. We improved the accuracy of BoDeGHa by using predictions of accounts in multiple repositories and we created an ensemble model based on multiple bot identification techniques. The results presented in this chapter are based on the following published peer-reviewed scientific articles and an associated tool that we developed:

- Golzadeh, M., Decan, A., & Mens, T. (2020). Evaluating a bot detection model on git commit messages. In 19th Belgium-Netherlands Software Evolution Workshop (BENEVOL)
  DOI 10.48550/arXiv.2103.11779

- BoDeGiC: An automated tool to identify bots in Git repositories by analysing commit information.
  https://github.com/mehdigolzadeh/BoDeGiC

- Golzadeh, M., Decan, A., Constantinou, E., & Mens, T. (2021a). Identifying bot activity in GitHub pull request and issue comments. In International ICSE Workshop on Bots in Software Engineering (BotSE).
  DOI 10.1109/BotSE52550.2021.00012

- Chidambaram, N., Decan, A., & Golzadeh, M. (2022). Leveraging predictions from multiple repositories to improve bot detection. In International ICSE Workshop on Bots in Software Engineering (BotSE).
  DOI 10.1145/3528228.3528403

- Golzadeh, M., Decan, A., & Chidambaram, N. (2022a). On the accuracy of bot identification tools. In International ICSE Workshop on Bots in Software Engineering (BotSE),
  DOI 10.1145/3528228.3528406

Fig. 1.1 (at the end of this chapter) provides an overview of the structure of the thesis chapters, highlighting the organization of the papers and their corresponding outputs.

Finally, *Chapter 7* highlights the contributions towards the thesis statement and reflects on the developed dataset, classification models and tools. It also opens perspectives and gives directions for future research.

1.10. Additional publications

Alongside the publications listed above, my PhD research has resulted in the following additional papers that are part of a warm-up in my PhD studies or that have resulted from collaboration with other researchers.

- Golzadeh, M., Decan, A., & Mens, T. (2019). On the effect of discussions on pull request decisions. In the 18th Belgium-Netherlands Software Evolution Workshop (BENEVOL),
http://ceur-ws.org/Vol-2605/16.pdf

- Golzadeh, M., (2019). Analysing socio-technical congruence in the package dependency network of Cargo. In the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering- Graduate student competition track (ESEC-FSE),
DOI 10.1145/3338906.3342497

- Decan, A., Mens, T., Rostami Mazrae, P, Golzadeh, M. (2022). On the Use of GitHub Actions in Software Development Repositories. In the 38th International Conference on Software Maintenance and Evolution (ICSME) 2022,
10.1109/ICSME55016.2022.00029

- Rostami Mazrae, P., Mens, T., Golzadeh, M. , Decan, A. (2023). On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. Empirical Software Engineering 2023,
10.1007/s10664-022-10285-5

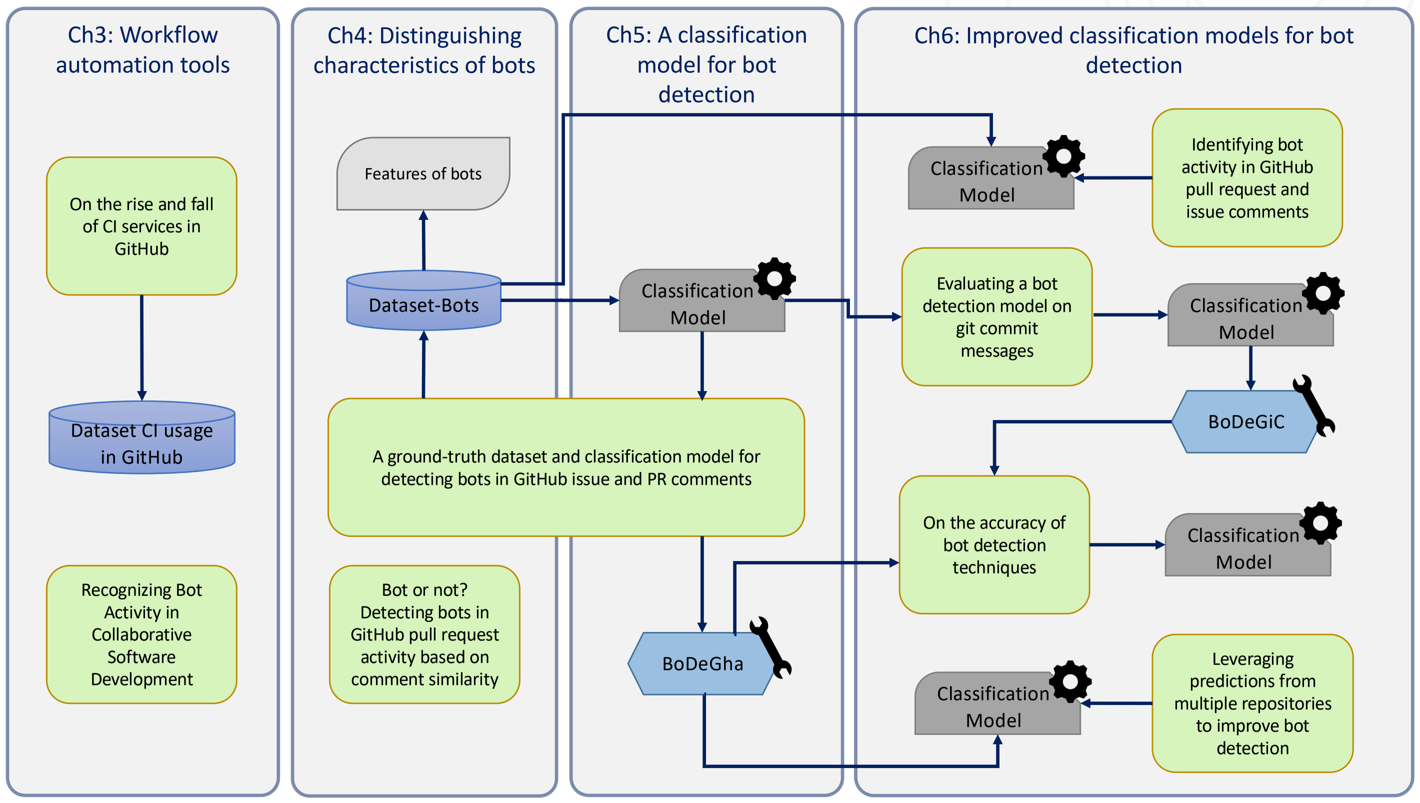

Ch3: Workflow automation tools
On the rise and fall of CI services in GitHub
Dataset CI usage in GitHub
Recognizing Bot Activity in Collaborative Software Development
Ch4: Distinguishing characteristics of bots
Features of bots
Dataset-Bots
A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments
Bot or not? Detecting bots in GitHub pull request activity based on comment similarity
Ch5: A classification model for bot detection
Classification Model
BoDeGha
Ch6: Improved classification models for bot detection
Classification Model
Identifying bot activity in GitHub pull request and issue comments
Evaluating a bot detection model on git commit messages
Classification Model
BoDeGiC
On the accuracy of bot detection techniques
Classification Model
Classification Model
Leveraging predictions from multiple repositories to improve bot detection

Figure 1.1: Structure of the dissertation chapters.

CHAPTER 2

State of the art

"Knowledge is the light that illuminates the path of those who seek it."

Mulla Sadra Mirdamad

The emergence of social coding platforms, the distributed collaboration in software development and the pull-based contribution style has changed the way developers collaborate in software projects, especially in open-source software development. Due to an increase in workload, workflow automation practices are becoming more popular. Developers widely use automation systems in order to support their repetitive tasks. For example, testing, building, quality checking, deploying code, detecting vulnerabilities and updating dependencies are no longer manual tasks and many software projects use CI tools and development bots to automate these repetitive tasks.

Numerous studies have been conducted to identify the positive and negative impact of workflow automation tools on software development processes. This chapter provides the state of the art of the research on this domain. It begins with a review of past and recent research about CI systems. It presents a comprehensive insight about the studies related to continuous integration and the impact of adopting such tools during collaborative software development. The chapter continues with studies that have focused on the use of bots in software engineering. The research in this area includes studies that propose bots to help software developers, studies that investigate the impact of bots in software development process and studies related to identifying bot account and bot activities.

2.1. Continuous integration systems

Continuous Integration (CI) is one of the core practices of the eXtreme Programming (XP) methodology proposed by Beck & Andres (2004). By making integration a daily practice, continuous integration aims to reduce the cost associated with integrating code developed by different developers within a team (or between different teams). While CI encourages collaborative code development and knowledge sharing its primary advantage is the reduced risk of large and cumbersome integrations (Beck & Andres, 2004).

With the rise of social coding platforms such as GitHub, software development has become increasingly collaborative. This has enhanced the popularity of modern distributed version control system (), especially those based on Git (Vasilescu et al., 2015b). Several CI services have been offered to support developers in social coding platforms (e.g., Travis, CircleCI, Azure DevOps and Jenkins) in order to ease the automation of build pipelines based on source code changes on VCS (Gousios et al., 2015). Studies indicate that CI is being adopted by more and more projects (Hilton et al., 2016a) and provide evidence of changes in the practice of these projects, such as an increase in commit frequency and an increase in test automation (Zhao et al., 2017).

Fowler & Foemmel (2006) presented ten core CI practices in which they claimed they could be successful in increasing the speed and improving the quality of software development feedback cycles. Elazhary et al. (2021) studied how organisations implemented these practices and found the benefits and challenges as well as how they changed the implementation based on their needs and preferences. Researchers have investigated different aspects of using CI services in projects, such as understanding the implementation of CI services in different environments (Ståhl & Bosch, 2014; Debbiche et al., 2014; Zampetti et al., 2017; Sampedro et al., 2018; Amrit & Meijberg, 2018), discovering their potential benefits and determining how to take advantage of them (Kaynak et al., 2019; Bernardo et al., 2018; Embury & Page, 2018; Rahman et al., 2018; Sampedro et al., 2018; Rahman & Roy, 2017), weaknesses (Rausch et al., 2017; Vassallo et al., 2019; Widder et al., 2019; Vassallo et al., 2018). Researchers also uncovered changes in the developer's behaviour after using a CI tool Gupta et al. (2017); Baltes et al. (2018); Zampetti et al. (2019) and analysed reasons and effects of build failure and success while using CI Jain et al. (2019); Vassallo et al. (2017); Rausch et al. (2017); Ortu et al. (2018). All these studies reveal the importance of CI tools and their immense impact on commercial and open-source software.

In addition, researchers classified software repositories utilizing CI tools according to their characteristics Gautam et al. (2017), the success of CI usage (Vasilescu et al., 2014), the human resources required to keep CI running (Manglaviti et al., 2017), analysed burdens of implementing CI (Mårtensson et al., 2017), build time of CI (Laukkanen & Mäntylä, 2015; Debroy et al., 2018; Rodríguez et al., 2013; Bisong et al., 2017; Ghaleb et al., 2019) and failure (Islam & Zibran, 2017; Paixão et al., 2017). Some others focused on continuous delivery (Paule et al., 2019; Laukkanen et al., 2018; Vassallo et al., 2016; Chen, 2015; Neely & Stolt, 2013) and deployment features of CI systems (De Jong & Van Deursen, 2015; Leppänen et al., 2015; Rahman et al., 2015; Claps et al., 2015; Rossi et al., 2016).

Vasilescu et al. (2015a) performed a quantitative study on 246 GitHub projects to demonstrate how CI can improve these projects. CI usage resulted in a higher volume of pull requests getting accepted, as well as a higher number of defects being discovered due to the increased volume of pull requests. In a 2016 study of 34,544 open-source projects on GitHub and a survey of 442 developers, many teams did not use continuous integration due to a lack of familiarity with the technology according to Hilton et al. (2016a). Yu et al. (2015) performed a regression analysis on 103K pull requests from 40 different GitHub projects in order to identify reasons that affect pull request evaluation latency. These studies focused on open-source software projects hosted on GitHub but there are also studies that have focused on CI usage in industrial cases.

In a series of studies Ståhl & Bosch (2013); Ståhl & Bosch (2014); Ståhl & Bosch (2016); Ståhl et al. (2017) analysed the relation between different aspects of software development and CI on industrial cases. Martensson et al. (2019) conducted a qualitative study with interviewees from four companies that develop large-scale embedded software systems. They built a model that shows how the CI/CD pipeline can be designed to include test activities. A mixed-method study conducted by Savor et al. (2016) of CI usage in two companies revealed that the CD part of the tools was not being utilized to its full potential due to company policies. Challenges identified with CI usage included: the need for continuous investment in the CI, the difficulty in finding and keeping skilled developers to configure and use the CI, the need to empower a culture that encourages developers to use the CI, and the additional effort involved to keep the software (including the CI) up to date.

Researchers studied available CI services such as Travis (Souza & Silva, 2017; Gallaba & McIntosh, 2020; Pinto et al., 2018), CircleCI (Gallaba et al., 2022), GitHub Actions (Chandrasekara & Herath, 2021; Valenzuela-Toledo & Bergel, 2022), and Azure (Buchanan et al., 2020). Beller et al. (2017) performed a quantitative study that examined the use of Travis in public GitHub repositories exclusively. They showed that, by 2017, Travis usage had increased a lot, being used in one-third of popular projects on GitHub. An analysis of 2.6M+ Java and Ruby builds revealed that CI usage was highly focused on testing-related tasks, mainly allowing developers to test their software on different platforms. It was, however, not possible for CI tools to replace local testing, due to the latency between writing the code and receiving feedback from the test. In another study, Widder et al. (2018) analysed 8,124 GitHub projects that had stopped using Travis. According to their findings, projects abandon this CI primarily due to its difficulty accommodating specific project complexity.

In a study conducted by Kinsman et al. (2021), researchers examined the impact of adopting GitHub Actions and observed that it increases the number of pull requests rejected and decreases the number of commits in merged pull requests. Their manual inspection of GitHub issues shows that developers have a positive perception of GitHub Actions. Valenzuela-Toledo & Bergel (2022) reviewed 222 commits that pertain to workflow changes and identified 11 types of changes. The authors identified several deficiencies in the production and maintenance of GitHub Actions workflow files, and recommended adequate tooling to facilitate the creation, editing, refactoring, and debugging of workflow files. Decan et al. (2022) analysed and characterised the use of GitHub Actions in nearly 70K GitHub repositories, of which 43.9% were using GitHub Actions workflows. They observed that workflows are predominantly utilized for development purposes, as with traditional CI tools and despite the fact that GitHub Actions could be used for large variety of automation purposes.

CI tools often serve as runners of bots, hence the presence of a bot often reflects the use of a CI tool in a project. Software developers frequently use bots to automate continuous integration reports and run tests and quality assurance tasks (Wessel et al., 2018b). Such bots are mainly responsible for notifying contributors of test failures in CI tools. Software engineering researchers oftentimes propose continuous integration bots in order to evaluate their new technique or approach to resolve problems in the software development process (Alizadeh et al., 2019; Hu & Gehringer, 2019; Ahlgren et al., 2021). Urli et al. (2018) focused on continuous integration builds and deployment aspects of using a bot. The authors proposed a bot that works as a CI build failure repair tool-chain for projects hosted on GitHub and that serves as a tool-chain to provide a program repair service. Danglot et al. (2019) proposed an approach that detects behavioral changes in commits and proposed a CI bot that detects behavioral changes in commits and proposes fixes to the developer to complement the commit.

2.2. Bots: evolution and origin

The first appearance of the term robot is credited to a 1921 science fiction play entitled Rossum's Universal Robots (Čapek, 1920), where the author replaced the classical term automaton with robot. Real mechanical robots only began to appear in the early 1970s (Kato, 1973; Mowforth & Bratko, 1987). The term *bot* (short for software bot) is an abbreviation of robot. However, unlike mechanical robots, bots are digital software experts. In the same way as robots perform tasks by manipulating the physical world, software bots act in the digital world, and are often used to automate repetitive tasks involving digital artifacts (Lebeuf et al., 2019). Software that is designed to behave like humans has been a goal since the dawn of computer science, not only to automate tasks we humans perform, but also to collaborate with humans on intellectual tasks that cannot be automated (Leonard, 1998).

The earliest idea of computer software imitating humans dates back to the ideas by Alan Turing in 1950 (Turing, 1950). Turing's test is considered by many to spark the idea of developing computer programs to interact with humans using natural language. Such programs are now commonly referred to as *chatbots* (Adamopoulou & Moussiades, 2020). Eliza was the first experience of conversing with a computer program, created by MIT professor Joseph Weizenbaum (1966). Since then, researchers have conducted many more studies on the interaction between computers and humans and created a wide verity of notable chatbots, including: (i) *PARRY*, that was introduced in 1972 by Kenneth Mark Colby, a psychiatrist and computer scientist at Stanford's Psychiatry Department (Colby et al., 1971); (ii) *Racter* (short for raconteur - a storyteller), written by William Chamberlain and Thomas Etter under the Inrac Corporation in 1983 (Chamberlain & Hall, 1984); (iii) *Dr. Sbaitso*, created by Creative Labs in 1992 as part of a Sound Blaster product called Sound Blasting Acting Intelligent Text to Speech Operator (Sbaitso, 1991); (iv) *TINYMUD*, a real–time multiplayer artificial player with a primary purpose of chatting (Mauldin, 1994); (v) *ALICE*, which stands for the Artificial Linguistic Internet Computer Entity, originated by Wallace (1995) (Shawar & Atwell, 2015); (vi) *Smarter-Child* was a real evolution in chatbot technology in 2001 as it could retrieve information from databases to help people with practical daily tasks (Molnár & Szüts, 2018).

Since then hundreds of different kind of software bots have been introduced by researchers and companies. Intelligent personal assistants have been developed by technological enterprises, such as IBM's Watson (High, 2012), Apple's Siri (Winarsky et al., 2012), Microsoft's Cortana (Levy, 2014), Amazon's Alexa (Lunden, 2014) and Google's Assistant (Statt, 2016). These personal assistants automate users' personal tasks through conversational interfaces.

Additionally, more narrowly focused bots have been developed and are widely used in different contexts. A plethora of studies have been published regarding their use (Motger et al., 2021; Akma et al., 2018; Ramesh et al., 2017). For example bots that are active in e-commence (Thomas, 2016; Lim et al., 2022; Oguntosin & Olomo, 2021), marketing (Rajaobelina & Ricard, 2021; Lin et al., 2022) and customer services (Cicco et al., 2022; Ben Mimoun et al., 2017; Thomas, 2016; Seiler & Schär, 2021; Haruna et al., 2021; Haugeland et al., 2022; Aattouri et al., 2021; Schanke et al., 2021; Go & Sundar, 2019; Følstad et al., 2018; Xu et al., 2017a). Researchers also conceptualize and analysed the use of chatbots for educational purposes (Li et al., 2021; Ranoliya et al., 2017; Bagmar et al., 2022). For example Molnár & Szüts (2018) discuss the issues of using chatbots as educational assistants, Colace et al. (2018) proposed a chatbot for e-learning. Chatbots are also used in other areas such as E-government and digital transformation (Cantador et al., 2021; Patsoulis et al., 2021; Reier Forradellas & Garay Gallastegui, 2021), in the healthcare sector and for medical purposes (Siedlikowski et al., 2021; Adikari et al., 2022; Minutolo et al., 2021; Sagstad et al., 2022; Fergencs & Meier, 2021; Horn et al., 2021), in the nutrition and food industry (Mendes Samagaio et al., 2021; Nalini et al., 2021) and even in the domains of cryptocurrency (Lee et al., 2021), Gaming (Allameh & Zaman, 2021) and human resources (Hillebrand & Johannsen, 2021).

Researchers have also investigated methods to improve the user experience in the use of chatbots (Gkinko et al., 2021; Pawlik, 2022; Schanke, 2021; Hildebrand et al., 2021; Brandtzæg & Følstad, 2017) and to benefit from more advanced technologies like Natural Language Processing (NLP) to enhance chatbots (Chao et al., 2021). One such technology is Generative Pre-trained Transformer (GPT), a state of the art NLP model developed by OpenAI, which has been shown to be effective in a wide range of NLP tasks including language generation, machine translation, and question answering (Brown et al., 2020). Such sophisticated bots are contributing human-readable content to peer production communities like Wikipedia (Cosley et al., 2007; Geiger, 2013; Geiger & Halfaker, 2017; Zheng et al., 2019; Kang et al., 2021) and Reddit (Long et al., 2017; Hurtado et al., 2019; Cruz, 2021). Bots are also widely used in social networks both for content production and governance (Aldayel & Magdy, 2022; Mendoza et al., 2020). But in social networks their usage is often abusive, as they are mostly aimed at spamming and sending fake news and hence they seek to hide their true nature. Because of this, numerous studies have focused on identifying these bots (Minnich et al., 2017; Efthimion et al., 2018; Rodríguez-Ruiz et al., 2020; Amleshwaram et al., 2013). Lebeuf et al. believe that technological advances, the mainstream adoption of text messaging and voice-based platforms, the move to service-oriented architectures, and the availability of public APIs and datasets are among the factors contributing to the widespread use of bots and the subsequent increase in new bots being designed, deployed, and used (Lebeuf et al., 2019).

This dissertation aims to analyze software development bots, also known as Devbots (Wessel et al., 2018a). They are designed to assist software developers when performing repetitive tasks, thereby relieving the workload of repository maintainers (Golzadeh et al., 2021). The number of such bots is increasing since the advent of social coding platforms. Devbots are active in every aspect of coding in collaborative development such as: ensuring license agreement signing, reporting continuous integration failures, analysing code quality, automated testing, security vulnerability analysis, dependency management, reviewing code, merging or closing pull requests, triaging issues, and refactoring the source code (Wessel et al., 2018a).

2.3. Bots: Definitions and classification schemes

Past research on software bots has led to diverse definitions and classification schemes (Lebeuf et al., 2019). Wessel (2021) provided a comprehensive overview of these definitions in her PhD dissertation. According to Wessel, researchers have defined software bots in terms of their specific usage scenarios:

Bots as automation providers. Several researchers have defined bots based on their ability to perform automated tasks. Storey & Zagalsky (2016); Cosley et al. (2007); Long et al. (2017) suggested that bots are programs that perform predefined repetitive tasks to increase the productivity of software developers. Wyrich & Bogner (2019) defined a bot as “intelligent software that acts autonomously toward achieving a defined goal and has the capability of interacting with another.”

Bots with conversational skills. Dale (2016) describes the relationship between humans and bots as “achieving some result through dialogic communication using natural language." Matthies et al. (2019); Abdellatif et al. (2020) defined bots as being able to communicate using human language. The term *chatbot* is the most frequently used term among researchers and practitioners to describe such conversational software bots. However, engaging in conversations is not necessary for software bots according to Lebeuf et al. (2017) and Paikari et al. (2019). In fact, chatbots are distinguished from other types of software bots by their ability to communicate with users through natural language.

Bots with human-like behaviors. Erlenhov et al. (2019) defined a DevBot in the context of software development as: "an artificial software developer which is autonomous, adaptive, and has technical as well as social competence." In the context of social media, Maus (2017) defines bots as "automated or largely automated programs that interface with online platforms in largely the same way that a typical human would be expected to: they hold normal accounts, make connections, and post content." Monperrus et al. (2019) claim that "we are now at the beginning of an exciting era where software bots will make contributions that are of similar nature than those by humans."

Bots as an interface between users and services. According to Lebeuf et al. (2019); Storey & Zagalsky (2016) a software bot is "an interface that connects users to services." This interface usually provides "additional value (in the form of interaction style, automation, anthropomorphism, etc.) on top of the software service's basic capabilities."

In a qualitative study, Erlenhov et al. (2020b) characterise and define from a practitioner's perspective software bots that are active in software development process. They systematically identify and categorise what qualities, characteristics, or properties turn a "Plain Old Development Tool" into a "bot" in the eyes of practitioners. They proposed 3 different personas for software development bots:

Chat bot persona: As the name suggests this definition covers tools that communicate with the developer through a natural language interface regardless of the tasks the bot is used for or how it actually implements the tasks. These kinds of bots are commonly an interface to an existing system and are not a system by themselves. Example of such bots are agents that communicate through comments with developers or bots that execute team or environment management tasks.

Autonomous bot persona: This bot persona emphasizes the highly independent nature of a bot. A bot is a tool that works autonomously (without much input from a developer) on a task that would normally be carried out by a human. These bots can sense automatically when their service is required after they are configured for a project. Therefore, not every scheduled script execution is an autonomous development, but a script that closes bugs or welcomes newcomers can be.

Smart bot persona: These bot personas are distinguished from plain old development tools by their smartness or ability to do sophisticated technical tasks. Such bots are less concerned about the ability to communicate than the ability to accomplish tasks in an unusually effective or adaptive manner. These bots are using relatively complex heuristics or are equipped with machine learning techniques. Agents that commit code and submit bug fixes are an example of such bots.

2.4. Software development bots

2.4.1 Bots in software engineering

Bots supporting software development activities have attracted the attention of software engineering researchers in the past few years. Researchers and practitioners use such bots in order to automate different activities in the software development workflow (Corti et al., 2019). In response to this emergence of bots, the *ICSE* conference has started to host workshops devoted to bots in software engineering [1]. The BotSE workshop series devoted to bots in software engineering is being held annually since 2019. This section presents an overview of the past and current research on software bots supporting software development activities.

Paikari & Van Der Hoek (2018) introduced a basic framework through which we examine the current state of chatbots in software development and identify directions for future work. Many studies have proposed conversational agents or chatbots to assist developers in the software development process, provide recommendations for developing such bots (Kuttal et al., 2020) and evaluate to what extent chatbots are suitable for collaborative modelling (Ren et al., 2020). To reduce the workload for documenting design decisions, Josephs et al. (2022) introduced a design decision bot for Slack and Black et al. (2019) benefited from automated speech recognition to create a bot for software modeling. Researchers have proposed chatbots to answer programming questions (Ilić et al., 2020; Xu et al., 2017b; Subramanian et al., 2019), to answer questions about API functions (Ed-douibi et al., 2020; Tian et al., 2017), to assist users in recognizing and clarifying technical details missed in queries to retrieve more relevant questions (Zhang et al., 2022a) and to answer software developers questions based on software crowdsourcing platforms (Ni et al., 2019). In the context of software testing Okanović et al. (2020) proposed a chatbot that walks developers through the process of properly specifying and performing a load test. Erlenhov et al. (2020a) conducted a study to identify

[1]www.botse.org

challenges in designing test cases for test bots.

Some researchers have implemented chatbots for assisting software engineering tasks (Chatterjee et al., 2021). For example Cerezo et al. (2019) implemented a chatbot for recommending experts, Qasse et al. (2021) proposed a chatbot for modeling and developing smart contracts, Paikari et al. (2019) developed a chatbot to detect and resolve potential code conflicts. Using more advanced technologies like NLP (Peérez-Soler et al., 2019; Matthies et al., 2019) and natural language understanding (NLUs) (Abdellatif et al., 2021) researchers have proposed more advanced software development chatbots.

Next to chatbots, other types of software bots are being used to automate repetitive activities in software development process. Researchers have proposed software bots for automatic classification of dialogue acts in bug repairing (Wood et al., 2020), domain modelling (Saini et al., 2020), facilitating information extraction from software repositories (Abdellatif et al., 2020) and recommending software engineering tools to software developers (Brown, 2019). Kumar et al. (2019) suggests using bots to extract data from software repositories, train models and provide information to engineers in order to increase the speed and efficiency of change and bug fixing. Beschastnikh et al. (2017) proposed a software analysis bot for research purposes that provides an interface for developers to subscribe to new research techniques without needing to trust the implementation.

Some software development bots submit pull requests or comments on submitted pull requests in order to suggest changes. Baudry et al. (2021) presented a bot for learning how to fix bugs based on continual training and applying changes by submitting pull requests. Rebai et al. (2019) propose a semi-automated refactoring documentation bot to recommend messages to document refactorings, their locations, and the quality improvement for a pull request if missing information is found. Wyrich & Bogner (2019) proposed a bot to automatically perform refactorings, resolve code smells and present the changes to a developer for asynchronous review via pull requests.

2.4.2 Bots in GitHub

In this dissertation we mainly focus on bots that are active in the GitHub social coding platform. They are called *GitHub bots* by Wessel (2021). A GitHub bot has a GitHub user profile like a human (or is associated with a GitHub app) and can act like a developer. They can open, close, and comment on pull requests and issues, do code reviews, commit code (Khanan et al., 2020). GitHub bots perform well-defined tasks within a development team to support other developers or act as an interface between developers and other tools such as CI tools. GitHub bots are *task-oriented bots* (Wessel, 2021) that connect user and services (Storey & Zagalsky, 2016), provide new forms of interactions with existing tools (Bradley et al., 2018) and integrate their work with human tasks (Farooq & Grudin, 2016).

Many studies have proposed new bots to automate tasks in collaborative development and reduce the workload of software repository maintainers. For example, Dominic et al. (2020) proposed a bot to welcome newcomers and onboard them. Serrano Alves et al. (2022) introduced a bot to suggest new tasks after onboarding the new user. Serban et al. (2021) and Carvalho et al. (2020) presented a bot that proposes fixes for static analysis warnings. Park et al. (2022) introduced an issue labeling bot classifying issue reports into custom labels. Mohayeji Nasrabadi et al. (2022) evaluated the impact of TODO bots on software development practices. A TODO Bot automatically creates a GitHub issue when TODO comments are added to a repository. Nakagawa et al. (2020) proposed a new clone modification support technique for integrating into pull request based development bot. da Silva et al. (2020) developed a virtual teammate to help developers and teams work on projects that highly rely on source control and issue tracking.

2.4.3 On the need for bot identification techniques

While the above examples reveal that bots play an important role in collaborative software development and in GitHub, bots also have downsides and may impose problems to repositories and their contributors or change the way they work. Therefore its important to identify the possible problems caused by bots, and provide guidelines to bot developers to avoid these problems (Wessel, 2020; Liu et al., 2020; Wessel et al., 2019; Pinheiro et al., 2019).

Studies have investigated how the use of bots changes the way maintainers work or improve the repository maintenance (Mirhosseini & Parnin, 2017; Wessel et al., 2018b; Erlenhov et al., 2022; Rombaut et al., 2023). Studies have also investigated how maintainers interact with human-created pull requests compared to those generated by bots (Wyrich et al., 2021) and pull requests without explanation (Monperrus, 2019). Wessel et al. (2020) studied how project maintainers experience code review bots, how a code review bot affects activity indicators in GitHub. In another study, they discuss six useful bots in GitHub's PR process Wessel & Steinmacher (2020). They analysed the negative aspects of bots in code contributions and introduce a meta-bot that acts as a middleman to mitigate this effect.

Despite the fact that bots can automate many tasks, they still need to interact with humans and deal with the resulting social and cognitive challenges (Brown & Parnin, 2019). Moharil et al. (2022) seek to understand how the adoption of a bot impacts the discussions in the issue-trackers of projects and Farah et al. (2022) investigate how users react to bot comments.

2.5. Bot identification

Although software bots are most often used for tasks that benefit their users in different ways, they can also be used for malevolent purposes. In social networks, bots are frequently spamming and sending fake news and hence they seek to hide their true nature. As a result, numerous studies have been conducted in order to identify them Minnich et al. (2017); Efthimion et al. (2018); Rodríguez-Ruiz et al. (2020); Amleshwaram et al. (2013). Wikipedia bots are active in several different tasks and make a huge amount of changes in the content of Wikipedia in different forms Geiger (2009). This enormous amount of activity can affect the entire Wikipedia ecosystem, so according to Zheng et al. (2019) they must be carefully evaluated and regarded in order to preserve the health of this online collaboration community.

While bots in social media are used for a variety of purposes, including malicious activities, and often attempt to hide their identity, bots in software engineering have a distinct nature, thus requiring different techniques for identification. As described in Section 2.4, bots play an important role in software engineering and are being used in different aspects of the software development process. While recent research regarding GitHub bots mostly focused on the practical value of bot adoption and proposing new bots, some studies investigated the impact of the presence of bots on software development workflows. To do so, they used very simple heuristics such as the presence of the string “bot” in the account name (Saadat et al., 2021) or relying on a list of bots (Wessel et al., 2018a) in order to identify bot accounts in the set of targeted repositories.

A prerequisite for correctly studying the impact of bots on software development processes is the ability to automatically identify such bots in a software repository. We found very few studies trying to identify bots. Wessel et al. (2018b) conducted a study about the prevalence and effect of bots in GitHub repositories. Erlenhov et al. (2019) presented a taxonomy that classifies 11 existing development-related bots in GitHub and Slack. Lebeuf (2018) provided a multi-faceted classification of bots (including many well-known examples of bots), combining their properties and behaviour. None of these studies proposes an automated approach to identify bots.

In parallel to my PhD research, Dey et al. (2020a) proposed an automatic method to identify bot accounts in Git projects. Each identity in their dataset consists of an author name and on email address. They studied three different approaches to find bots, based on (i) the presence of the string "bot" at the end of the author name, (ii) commit messages, and (iii) features related to files changed in commits and projects the commits are associated with. They combined these three different approaches into a single ensemble bot identification model that was validated on a dataset of 67 bots of which 58 cases (85%) were effectively captured by the model.

The study of Dey et al. (2020a) has some limitations that we will address in this dissertation. One limitation is that their dataset is based only on commit data in GitHub repositories. We found many examples of bots that are not involved in commit activity but only involved in issue and PR activities. In Chapters 4, 5 and 6 of this dissertation we will provide classification techniques for bot identification that cover other types of GitHub bots.

CHAPTER 3

Workflow automation tools

“Through knowledge, we can transform ourselves and the world around us.”

Sheykh Bahai

Workflow automation tools such as continuous integration systems and bots are abundant in social coding platforms such as GitHub. Continuous integration, delivery and deployment (CI) tools automate the integration of code changes from multiple contributors into a central repository where automated builds, tests and code quality checks run. This chapter presents the results of an empirical study that we have conducted on the use of CI tools in a dataset of GitHub repositories containing the development history of npm packages. We observe that more than half of the active repositories in this dataset use at least one CI tool. We investigate the evolution of CI usage over time as well as the simultaneous usage and migration among different available CI tools.

Bots are another type of cloud-based automation in social coding platforms. They are used to reduce the workload of software developers by performing repetitive tasks on their behalf. We empirically analyse the presence of bots among the active contributors of GitHub repositories. We show that although some of these bots are labeled as such by GitHub, most of them use normal accounts and their activities are not tagged as belonging to bots. We explain the implications of not signaling the presence of bots and highlight the importance of identifying bots from both a practitioner's and a researcher's point of view.

3.1. On the prevalence of CI in GitHub

Continuous integration, deployment and delivery have become the cornerstone of collaborative software development and DevOps practices. CI automates the integration of code changes from multiple contributors into a central repository where automated builds, tests and code quality checks run. Well-known examples of CI services are Jenkins, Travis, CircleCI and AppVeyor. CI services can also be built-in in social coding platforms such as GitHub and GitLab (Dabbish et al., 2012). GitLab already featured CI capabilities since November 2012. Based on popular demand, and in response to CI support integrated in GitLab, GitHub publicly announced the beta version of GitHub Actions (GHA) (abbreviated to GHA in the remainder of this dissertation) in October 2018. In August 2019, they officially began supporting CI through GHA, and the product was released publicly in November 2019.

GHA allows to automate a wide range of tasks based on a variety of triggers such as commits, issues, pull requests, comments and many more (Chandrasekara & Herath, 2021). GHA can be used to facilitate code reviews, code quality analysis, communication, dependency and security monitoring and management, testing, etc. GHA facilitates the integration with external services, and can even obviate the need of using such external services altogether.

GitHub is by far the largest social coding platform, hosting the development history of millions of collaborative software repositories, and accommodating over 56 million users in September 2020 (GitHub, 2020). Given its popularity and the ease with which GHA allows to automate the CI workflow, we hypothesise that GHA has had a significant impact on today's CI landscape. More particularly, we believe that it has increased the awareness of the need for CI, it has reduced the entry barrier for projects to start using CI, and it may have lead projects to migrate from other CI services towards GHA.

This chapter aims to quantitatively and impartially verify these hypotheses, and discusses their consequences, through a longitudinal analysis of how different CIs have been used over a nine-year period in 91,810 GitHub repositories corresponding to the software development history of reusable Node.JS packages distributed through the npm package registry. The following four sections explore these research questions:

RQ_1 How did the CI landscape evolve? We identified 20 different CIs being used in the considered set of repositories, some of which were considerably more prevalent than others. Together with Travis, GHA covers more than 80% of all usages. Moreover, in only 18 months GHA has overtaken all other

CIs in popularity.
RQ_2 *What are the most frequent combinations of CIs?* We observed that many repositories have used multiple CIs during their lifetime. AppVeyor is nearly always used in combination with some other CI. If a repository uses a CI simultaneously with another one, it is mostly in combination with Travis, GHA or CircleCI.
RQ_3 *How frequently are CIs being replaced by an alternative?* We observed a non-negligible amount of CI migrations. GHA attracted most of these migrations. The majority of migrations were moving away from Travis and towards GHA.
RQ_4 *How has the CI landscape changed since GHA was introduced?* Based on a regression discontinuity design, we found that the usage of Travis, Azure and CircleCI has been negatively affected by the introduction of GHA.

3.1.1 Data extraction

In order to analyse the use of CIs in software development repositories on GitHub, we need a large dataset containing thousands of GitHub repositories for a wide range of software programming projects serving different purposes and exhibiting variations in longevity and size. The dataset should exclude repositories that have been created merely for experimental or personal reasons, or that only show sporadic traces of commit activity (Kalliamvakou et al., 2014b). Registries of reusable software packages (e.g., npm for JavaScript or Maven for Java) are good candidates to find such large datasets, as they typically host thousands of software packages at different levels of maturity and popularity. However, not all packages belonging to such registries have an associated Git repository on GitHub.

According to the libraries.io open source monitoring service, npm is by far the largest of all listed package registries (Katz, 2020). We used the public API of npm to list all 1.6M+ scoped packages. We downloaded the metadata of each package on 23 May 2021 and found that 803K packages mention an associated git repository hosted on GitHub. We cloned 676K of these repositories, the remaining ones corresponding to repositories that were no longer available. Since one of our goals is to study how the CIv landscape has evolved these recent years, we excluded repositories that were not *active* during the last year of the observation period (i.e., they had no commit between 24 May 2020 and 23 May 2021). We also excluded 11,557 forks, since part of their history duplicates the one of the forked repository. This left us with 201,403 repositories.

CI usage in a repository is typically visible through the presence of some specific CI-related configuration files. For example, the presence of a .travis.y(a)ml file indicates that Travis CICI has been configured for this repository while a .y(a)ml file in the .github/workflow/ folder triggers GHA. Based on an exploration of scientific publications, blog posts and developer websites, we established an initial list of 28 candidate CI tools and services to consider. We carefully went through their documentation to determine the file paths that must be considered to detect the presence of each CI. We excluded CI that are mostly configured through a dedicated UI and not a configuration file, or CIs whose configuration file cannot be detected (e.g., because the file path and file name can be freely chosen by the users). After such exclusion 20 CIs remained: Travis, GHA, CircleCI, AppVeyor, Azure, Jenkins, GitLab CI, Drone CI, Hound CI, Bitbucket CI, Wercker, Golang CI, CodeBuild, Buildkite, Semaphore, Cirrus, CloudBees, Amplify, Buddy and Bitrise.

We then checked every commit of all 201,403 cloned repositories for the presence of CI-related configuration files. We found 179,535 distinct CI-related file paths in 95,035 repositories. We relied on these file paths as a proxy to detect if and when a CI was used by a repository.

Working with Git histories can be quite challenging (Bird et al., 2009) and leads to a range of data quality issues that need to be dealt with. A first issue is that Git is revisionist, allowing one to rewrite the history of a repository. Unfortunately, there is no way to detect such rewritings, except when it leads to inconsistencies (e.g., commits referring to files that were not yet created or that were already removed, commits with invalid dates, etc.). We identified and removed 60 repositories for which we found such inconsistencies for CI-related files. We contacted the maintainers of two of such repositories, who confirmed the inconsistencies, explaining that it was the consequence of an earlier migration to Git and a merge of different repositories into a single one.

Another issue stems from the presence of implicit branches and the fact that the history of a Git repository is represented by a directed acyclic graph, as opposed to a tree-like structure in svn. As a result, chronological sequences of commits may originate from distinct branches, and may contain *a priori* contradictory changes (e.g., a file that is added and removed multiple times over relatively short periods of time). Since we determine the use of a CI based on the presence of specific file paths, we are particularly exposed to this issue. We found several cases of chronological sequences of CI-related files being added and removed multiple times in a row. Since such file removals did not correspond to a deliberate intent to remove the corresponding CI service, we ignored all removals of CI-related files for which the same file was found to be reintroduced within 30 days. We empirically found that this value allows to capture 98.1% of the cases we found, while preserving actual cases of re-introductions of a CI.

After this data cleaning step, we used the presence of CI-related files to determine when a CI was added and (possibly) removed from a project. One should note that a particular CI can be added and removed more than once during a project's lifetime. By manually inspecting CI usages in repositories, we noticed several cases of repositories experimenting with the integration of a CI for a few days only. We excluded such cases by removing 6,910 usages (found in 6,586 repositories) whose duration did not exceed 30 days. We also found and excluded 3 cases of repositories making use of a high number of different CIs at the same time. We manually inspected these repositories and found that their purpose was to showcase integration of an app with various CIs. An example of such a repository is cypress-io/cypress-example-kitchensink.

The final dataset contains 119,033 CI usages spread across 91,810 repositories. The higher number of usages than repositories signals that there are many repositories that use multiple CIs. Table 3.3 at the end of this chapter provides an overview of the CIs found in repositories, along with the number of usages and the number of repositories in which they were found at least once through time. Travis and GHA are by far the most popular CIs, being used each by more than half of the repositories with a CI and together covering more than 90% of all repositories with a CI. This may be expected for Travis, since it has existed since 2011 and thus repositories have had more time to use this service. It is surprising to find GHA as the secondmost popular CI, given that its first usage is observed in 2019 only, making it the most recent of the considered CIs. The five next popular CIs, following at quite a distance, are CircleCI, AppVeyor, Azure, GitLab CI and Jenkins. The remainder of this chapter exclusively focuses on these seven CIs since, together, they represent 99% of all CI usages, and cover 99.6% (91,426) of all repositories having used a CI.

The data and code to replicate the analysis in this chapter are available on `https://doi.org/10.5281/zenodo.5815352`.

3.1.2 How did the CI landscape evolve?

RQ_1 is exploratory in nature, aiming to obtain a better understanding of which CIs are found in the repositories of our dataset, how prevalent these CIs are, and how this has been changing over time. This will provide the necessary context to interpret the results of the subsequent research questions.

Obviously, not all repositories make use of a CI. Fig. 3.1 shows the evolution

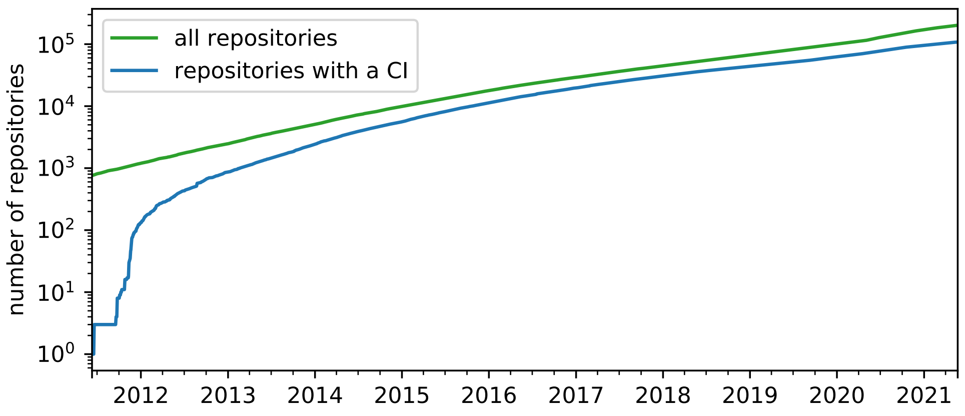


Figure 3.1: Evolution of the number of repositories (green line) and number of repositories using a CI (blue line).

of the number of repositories in our dataset that use a CI compared against the total number of repositories we considered at that time. We observe that the number of repositories using a CI has a tendency to increase over time. Proportionally to the number of considered repositories, it went from 56.7% in January 2015 to 63% in January 2020. At the end of the observation period (i.e., in May 2021), the proportion of repositories with a CI is 53.6%. This lower proportion is likely to be a consequence of the fact that recent repositories did not have enough time yet to adopt a CI. According to Hilton et al. (Hilton et al., 2016b), the median time to adopt a CI is one year.

These observations are in line with the ones of Hilton et al. (Hilton et al., 2016b), and confirm that CIs are widely used in practice nowadays. However, that does not mean that all CIs are equally used. Fig. 3.2 breaks down the blue curve of Fig. 3.1 for the seven most popular CIs, showing the evolution of the number of repositories using each CI. Since the number of repositories using Travis and GHA has a different order of magnitude than for the other CIs, we used a different y-axis scale starting from 10,000 repositories (illustrated by the horizontal dashed line).

We observe that Travis, the oldest CI has dominated the landscape since its introduction in 2011. This is not surprising since Travis was the only available CI for the first two years of the observation period (at least in our dataset). This explains why Travis accounts for the highest number and proportion of repositories in Table 3.3. Other popular CIs have entered the CI landscape only more recently: CircleCI and AppVeyor emerged in 2014, GitLab CI in 2015,

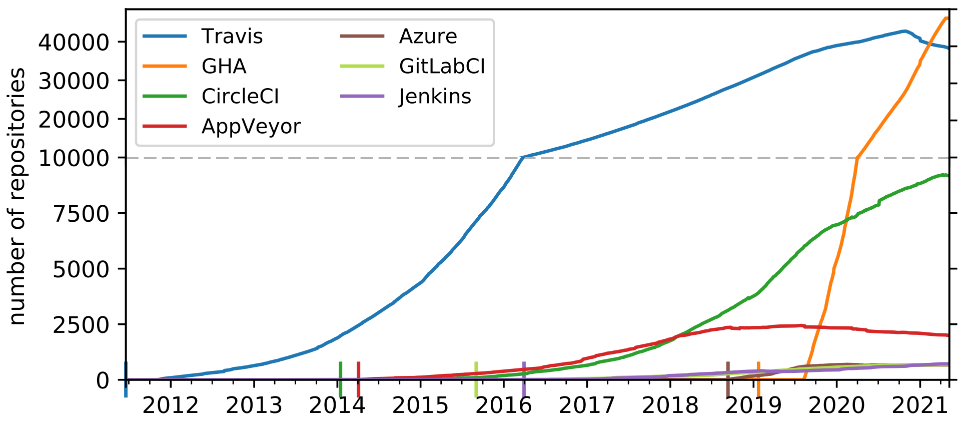


Figure 3.2: Number of repositories using a specific CI.

and Jenkins in 2018. Since their introduction in the first half of 2014, CircleCI and AppVeyor were successful in attracting developers and obtained a good share of repositories using these CIs. Even GitLab CI, originally introduced as an integrated CI service for GitLab, started to attract a small share of GitHub repositories since September 2015. Although Jenkins was one of the earlier CI tools for Java applications, it only shows up in our data as of April 2016, and it has a small share of repositories using it. This can be explained by two phenomena. First of all, our dataset focuses on npm packages, hence they are less likely to contain Java development. Secondly, the *pipelines-as-code* feature[1] only became available in Jenkins in April 2016, implying that we were only able to track repositories using this feature as of that date. Finally, we found evidence of a rapid growth of repositories using Azure following its introduction in 2018, but after a short period of time the number of repositories using Azure stagnated.

Probably the most surprising phenomenon that can be observed in Fig. 3.2 is the very rapid growth of GHA usage. Despite the fact that GHA was only introduced in 2019, it has become the second most popular CI in our dataset just 18 months after its introduction, even overtaking the dominant position of Travis! At the end of the observation period, GHA covers 51.7% of the repositories with a CI, followed by Travis (42.5%), CircleCI (10.2%) and AppVeyor (2.2%). The remaining CIs together (Azure, GitLab CI and Jenkins) do not exceed 2.3%. This suggests that GHA has changed the CI landscape dramatically, fuelling the need to carry out a deeper empirical analysis.

[1] `https://developers.redhat.com/blog/2016/08/24/whats-new-in-jenkins-2-0-2`

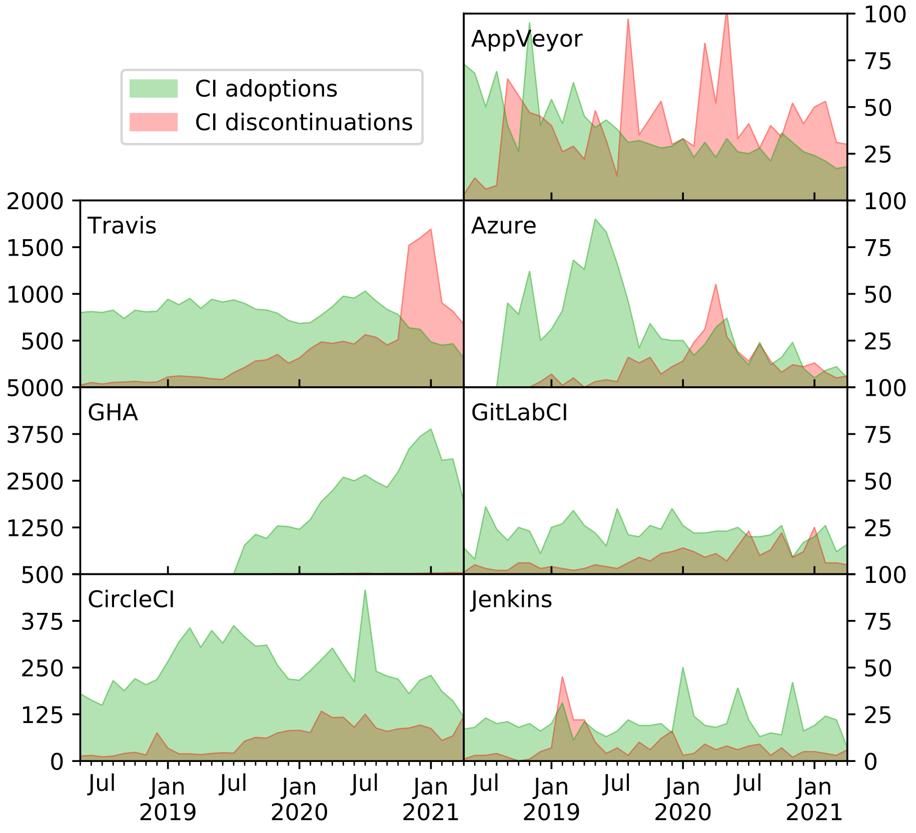


Figure 3.3: Monthly number of repositories adopting and discontinuing a CI.

From Fig. 3.2, we observed for most CIs either a slight recent decrease in the number of repositories using it (Travis and AppVeyor) or a reduced growth in this number (CircleCI, Azure and GitLab CI). These observations can be the consequence of less repositories adopting the CIs, of more repositories discontinuing them, or a combination of both. Fig. 3.3 shows the monthly churn in CI usage for the last three years of the observation period, computed as the monthly number of repositories that adopted and discontinued each CI.

We observe that the recent decrease in the number of repositories using Travis is due to a combination of less repositories adopting Travis (from July 2020 onwards) and of many repositories that discontinued using them. This is especially visible from late 2020 onwards, where the monthly number of repositories that discontinued using Travis went from 508 in October to 1,520 in November 2020.

We also observe that the reduced growth of CircleCI, AppVeyor and Azure is mostly a consequence of less repositories adopting these CIs. For instance, the number of monthly adoptions of CircleCI went from an annual average of 308 in 2019 to 253 in 2020, and only 173 in 2021. Similarly, the number of adoptions of Azure dropped from 90 in May 2019 to only 17 in February 2020, reaching the number of discontinuations. In AppVeyor, there are even more discontinuations than adoptions since July 2019.

Interestingly, most of the churn we observed for these CIs started a few months after the introduction of GHA. GHA is the only considered CI that exhibits a steadily increasing adoption rate, and nearly no discontinuations (only 374 compared to 46,416 adoptions during the observation period). We will therefore analyse and discuss the impact of GHA on the CI landscape in detail in RQ_4.

3.1.3 What are the most frequent combinations of CIs?

Table 3.3 revealed that there are more CI usages than repositories. This implies that some repositories use more than one CI (either simultaneously or at different points in time). Table 3.1 shows the number of repositories in function of the number of distinct CIs they have used during their observed lifetime. While three out of four repositories have only used a single CI throughout their lifetime, there is still a large proportion that have used 2 different CIs (21.7%) or even more than two (3.3%). This confirms that it is not unusual for a repository to use multiple CIs throughout its lifetime.

Fig. 3.4 shows the proportion of repositories having used a given pair of CIs A and B. Since we found that most repositories having used multiple CIs involved Travis (86.2%), GHA (83.7%) and CircleCI (22.3%), and to avoid the analysis to be biased by these most frequently used CIs, we computed the proportions relative to the number of repositories having used a given CI

Table 3.1: Number and proportion of repositories having used different CIs during their lifetime.

# CIs →	**1**	**2**	**3**	**4**	**5**
# repositories	68,549	19,799	2,647	371	60
% repositories	75.0%	21.7%	2.9%	0.4%	0.07%

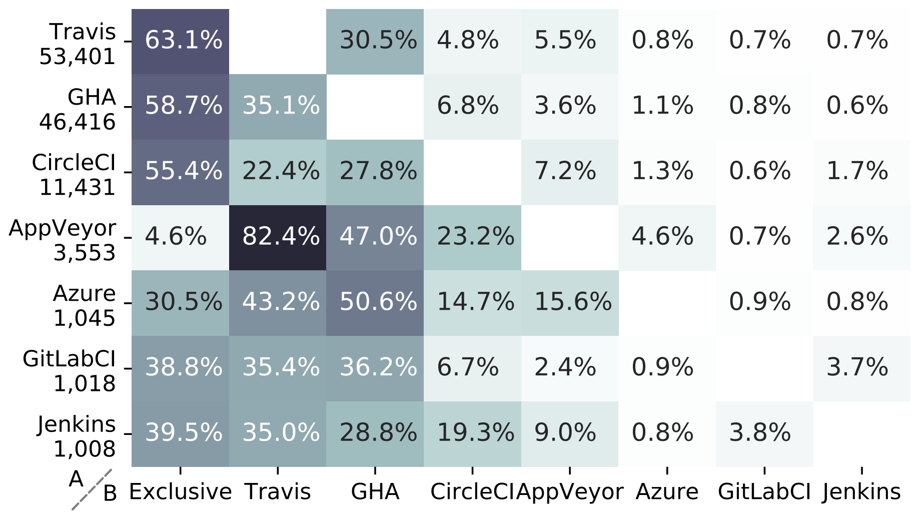


Figure 3.4: Proportion of repositories having used CIs A and B relative to all repositories having used A.

(this is, the proportion of repositories having used CIs A and B relative to all repositories having used A). For example, 30.5% of the repositories with Travis have used GHA, 5.5% have used AppVeyor and 4.8% CircleCI. Note that the sum may exceed 100% (e.g., for AppVeyor) since repositories may have used more than two CIs.

We also report in the first column of Fig. 3.4 the proportion of repositories in which a CI has been exclusively used. We observe that a majority of the repositories have exclusively used Travis (63.1%), GHA (58.7%) or CircleCI (55.4%). The opposite can be observed for AppVeyor, Azure, GitLab CI and Jenkins. Out of the 1,045 repositories having used Azure, more than half of them (50.6%) also have used GHA. This is even more pronounced for AppVeyor: only 4.6% of the 3,553 repositories have used it exclusively, while 82.4% of the repositories have also used Travis, 47.0% have also used GHA, and 23.2% have also used CircleCI.

So far, we considered pairs of CIs, disregarding whether the CIs were used *simultaneously*. Let us therefore consider the ***co-usage*** of CI pairs in a given repository, defined as those situations where the repository uses both CIs for a common period of at least 30 days. If a repository would simultaneously use 3 different CIs A, B and C, this will be counted as 3 co-usages, namely for each pair (A, B), (B, C) and (A, C).

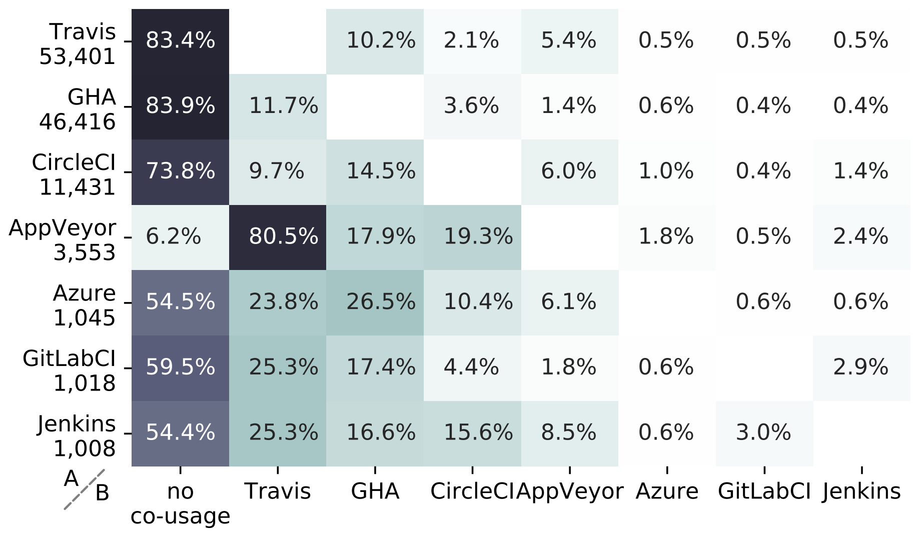


Figure 3.5: Proportion of co-usages of CIs A and B relative to all repositories using A.

We found 14,335 co-usages in 11,049 distinct repositories out of 22,877 repositories having used multiple CIs during their lifetime. Travis, AppVeyor, GHA and CircleCI are involved in most co-usages, together covering 86.7% of all co-usages and 92.1% of all repositories with co-usage.

Fig. 3.5 shows the proportion of co-usages we found for each pair of CIs A and B, relative to all repositories involving A. We also report in the first column the proportion of cases with no co-usage. For all CIs except AppVeyor, more than half of the repositories using them do not use another CI at the same time. Travis, GHA and CircleCI are considerably less frequently used in combination with another CI than the others CIs. At the other extreme, only 6.2% of the usages of AppVeyor do not involve another CI at the same time.

Focusing on the actual cases of co-usage (i.e., all but the first column), AppVeyor is mostly co-used with Travis (80.5%), and to a lesser extent with CircleCI (19.3%) and GHA (17.9%). Travis is the most frequent complementary CI for GHA, AppVeyor, GitLab CI and Jenkins, and GHA is most frequently complemented by Travis, CircleCI and Azure.

3.1.4 How frequently are CIs being replaced by an alternative?

The findings of RQ_2 revealed a considerable amount of combinations of multiple CIs within repositories. Part of these combinations were due to ***co-usages***. Another likely scenario is that a repository has used multiple CIs during its lifetime, but not necessarily simultaneously. This could be the case, for example, when a repository maintainer is unsatisfied with its current CI and decides to replace it with another CI that offers solutions that better meet the needs of the team or the project.

We consider that a repository ***migrated*** from some CI A to another CI B if the repository stopped using A and started using B at "around the same time", i.e., within a time window of 30 days either *before* or *after* B started to be used by the repository. The rationale behind this time window is to accommodate for a possible transition period during which either both CIs were being used together (i.e., time needed to remove A after the introduction of B) or where none of them was being used (i.e., time needed to introduce B after the removal of A).

We detected 14,219 such cases of migration in 13,083 different repositories. Table 3.2 reports on the number of repositories involved in a migration away from or towards a given CI. It also reports on their proportion relative to the total number of repositories using a given CI. We observe that most migrations (11K+) are due to repositories migrating away from Travis and, to a lesser extent, from CircleCI. On the other hand, we see that most migrations (12K+) target GHA and, to a lesser extent, again CircleCI. For most of the CIs, we observe there are more migrations away from them than migrations towards them, with the notable exceptions of GHA that attracted far more repositories, and of CircleCI and Azure, both having roughly the same number of repositories that migrated from and to them. Looking at the proportions of repositories relative to the total number of repositories using a given CI, we see that around one out of five repositories using Travis, AppVeyor, Azure or GitLab CI migrated to another CI. On the other hand, only 29 out of the 46,416 repositories using GHA migrated away from it. The last column reveals that migrations explain more than one out of four repositories using GHA or Azure. In contrast, less than 1% of the repositories using Travis or AppVeyor are due to a migration.

Going into the details of these migrations between CIs, Fig. 3.6 shows the proportion of migrations from CI A to CI B relative to the total number of migrations away from A. This allows to see the distributions of the target

Table 3.2: Number of repositories that migrated away from and towards a CI, and their relative proportion.

		migrated away from		migrated towards	
CI	**repo.**	#	%	#	%
Travis	53,401	11,591	21.7%	218	0.4%
GHA	46,416	29	0.06%	12,269	26.4%
CircleCI	11,431	1,382	12.1%	1,323	11.6%
AppVeyor	3,553	698	19.6%	30	0.8%
Azure	1,045	222	21.2%	282	27.0%
GitLab CI	1,018	192	18.7%	53	5.2%
Jenkins	1,008	105	10.4%	44	4.4%

CIs for migrations originating from A. We observe that GHA proportionally represents the vast majority of the targets of a migration, regardless of the considered source CI. GHA attracted up to 96.8% of all migrations away from Azure. The only notable exception is Jenkins: even if GHA still accounts for 41.0% of the migrations from Jenkins, more than half of the migrations are towards Travis and CircleCI. Travis is also the second most frequent target for migrations originating from CircleCI and GitLab CI.

Fig. 3.6 provided insights about the *target* of CI migrations. To provide insights about the *source* of CI migrations, Fig. 3.7 reports on the proportion of migrations from CI A to CI B, this time relative to the total number of migrations towards B. We observe that Travis provided the overwhelming majority of the repositories that migrated to another CI, regardless of the target CI. It represents up to 93.0% of all migrations towards CircleCI, accounting for 1,230 migrations. On the other hand, even if CircleCI accounts for more than half of the migrations towards Travis, this corresponds to 119 migrations only. CircleCI is also the secondmost frequent source of migrations for all CIs, except for Azure whose secondmost frequent source of migrations is AppVeyor. Nearly one third of all migrations towards Azure originate from AppVeyor.

3.1.5 How has the CI landscape changed since GitHub Actions was introduced?

RQ_1 revealed that, since the public release of GHA in November 2019, its market share has been increasing very fast, becoming the dominating CI in less than 18 months time! GHA hence seems to have played a major role in

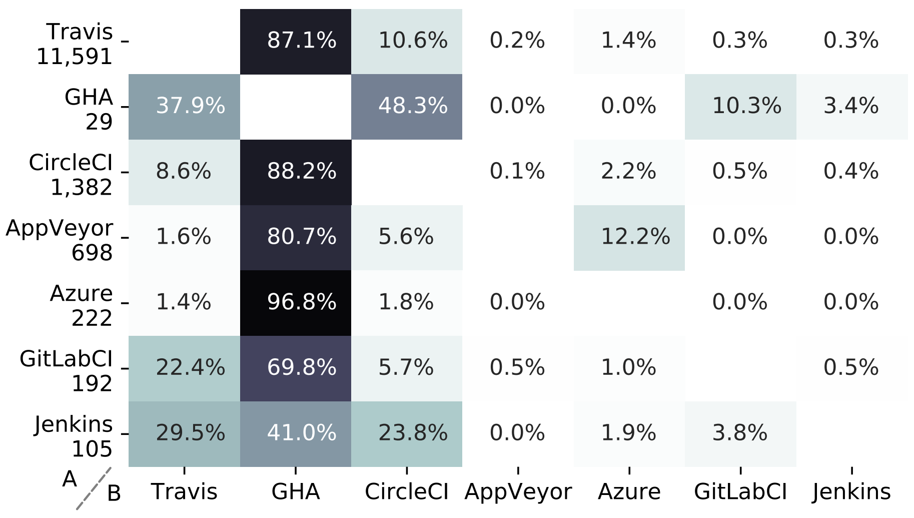
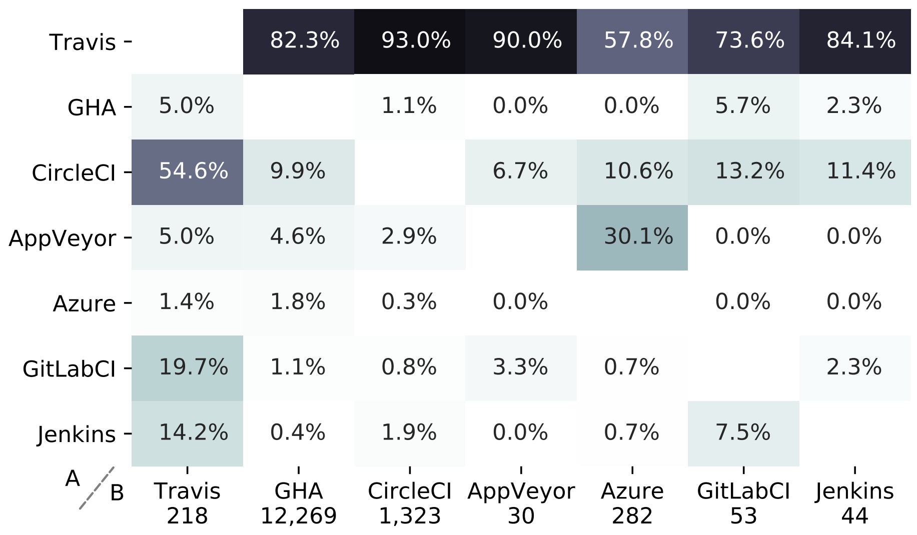


Figure 3.6: Proportion of migrations from CI A to CI B, relative to the total number of migrations away from A.

how the CI landscape had changed, with many repositories co-using GHA with some other CI (RQ_2), or even migrating towards GHA (RQ_3). The increase in popularity of GHA seems to have had a diminishing effect on the share of repositories using Travis and other CIs. RQ_4 therefore aims to scrutinise to which extent GHA has altered the CI landscape.

To study the effect of the introduction of GHA on the usage of other CIs, we use the statistical technique of (linear) Regression Discontinuity Design (RDD) (Thistlethwaite & Campbell, 1960; Cook & Campbell, 1979). This technique allows to model the effect of a particular important event by comparing the situation during a given time window before and after the event. In our case, we intend to use RDD to model the effect of the introduction of GHA on CI usage of repositories before and after this event. We consider August 8, 2019 as the event date since it corresponds to the day at which GHA announced the availability of a CI/CD service. RDD assumes that one would be able to observe a discontinuity in the data if the event affects the outcome (here, CI usage). Such a discontinuity would reveal itself as a perceptible difference in the intercept and/or slope of the (linear) regression model before and after the event. The RDD model is formulated as:

$$y_i = \alpha + \beta \times \text{time}_i + \gamma \times \text{event}_i + \sigma \times \text{time\_after}_i + \epsilon_i$$



A / B	Travis 218	GHA 12,269	CircleCI 1,323	AppVeyor 30	Azure 282	GitLabCI 53	Jenkins 44
Travis		82.3%	93.0%	90.0%	57.8%	73.6%	84.1%
GHA	5.0%		1.1%	0.0%	0.0%	5.7%	2.3%
CircleCI	54.6%	9.9%		6.7%	10.6%	13.2%	11.4%
AppVeyor	5.0%	4.6%	2.9%		30.1%	0.0%	0.0%
Azure	1.4%	1.8%	0.3%	0.0%		0.0%	0.0%
GitLabCI	19.7%	1.1%	0.8%	3.3%	0.7%		2.3%
Jenkins	14.2%	0.4%	1.9%	0.0%	0.7%	7.5%	

Figure 3.7: Proportion of migrations from CI A to CI B, relative to the total number of migrations towards B.

where i corresponds to an observation related to a given CI before and after the event. In order to incorporate the passage of time into the model, three parameters are used: $time_i$, $event_i$ and $time\_after_i$. The *time* parameter is the number of months that have passed from the beginning of the observation window. We use two observation windows of 12 months each, respectively until 1 month before and starting from 1 month after the event. We purposefully ignore what happened between 30 days before and 30 days after the event to account for possible instabilities close to the event date. The binary *event* parameter specifies whether an observation is measured before (event $= 0$) or after the event (event $= 1$). The *time_after* parameter indicates elapsed time (expressed in number of months in our case) since the event and it is set to 0 before the event. ϵ_i is the residual error. The two resulting linear regression lines have a slope of β before the event, and $\beta + \gamma$ after it. The difference between the two regression values of y_i indicates the size of the effect of the event. The accuracy of the RDD model is estimated using R^2, a common method for assessing the goodness of fit of a regression model. We implemented the RDD model based on the ordinary least square method using the statsmodels library in Python.

Given our aim to determine the effect of the public release of GHA on the usage of other CIs, we computed for each CI the monthly variation of CI usage, measured as the number of repositories adopting the CI minus the number of repositories discontinuing the CI. We fit 7 different RDD models: one for assessing the global effect of the event on the CI landscape, and one for the individual effect on each of the 6 most popular CIs (excluding GHA itself). The results are reported in Table 3.4 (at the end of this chapter). We provide the R^2 goodness of fit of each model, the coefficients of each model parameter, and the statistical significance of the coefficients in terms of their p-value. We consider three levels of significance, reflecting the strength of the effect induced by the event: $p < 0.001$ for strongly significant (***), $p < 0.01$ for highly significant (**), and $p < 0.05$ for moderately significant (*).

Only 4 out of the 7 computed RDD models provide a sufficiently high goodness of fit: CI landscape, Travis, Azure and CircleCI (in decreasing order of goodness of fit, with R^2 ranging from 0.94 to 0.57). We will therefore only discuss these 4 models in more detail. Only for Travis ($p < 0.001$), Azure ($p < 0.001$) and CircleCI ($p < 0.01$) we observe a statistically significant effect of the event. The coefficients of *time* and *time_ after* are significant for CircleCI and Azure. These coefficients indicate that the introduction of GHA had a significant impact on Travis, Azure and CircleCI usage.

To observe the effects of the introduction of GHA, Fig. 3.8 provides a visualisation of the RDD models for Travis, CircleCI and Azure. The figure confirms the observations of the statistical analysis, revealing a decline in the monthly variation of CI usage for each of them. We also observe a change in the direction of the slope (from positive to negative), suggesting that after the event the growth rate of CI usage decreases. Still, for Travis and CircleCI, the number of adoptions remains higher than the number of discontinuations. For Azure, however, the monthly CI usage variation start to become negative after the event, implying that more repositories discontinue Azure compared to those repositories adopting it.

In the realm of software development, the utilization of CI tools as runners of bots is quite common, with the presence of a bot often indicative of the employment of a CI tool in a project. These bots, which can be seen as the visible manifestations of automation within the software development process, serve as essential components in automating various tasks. They play a crucial role in streamlining the CI workflow by automating tasks such as generating reports, running tests, and ensuring quality assurance (Wessel et al., 2018b).

Notably, the presence of bots within a project signifies their active involvement in executing these vital functions. Bots can be considered as the tip of the iceberg when it comes to automation in the software development process. They represent the visible and tangible outcomes of the underlying automation

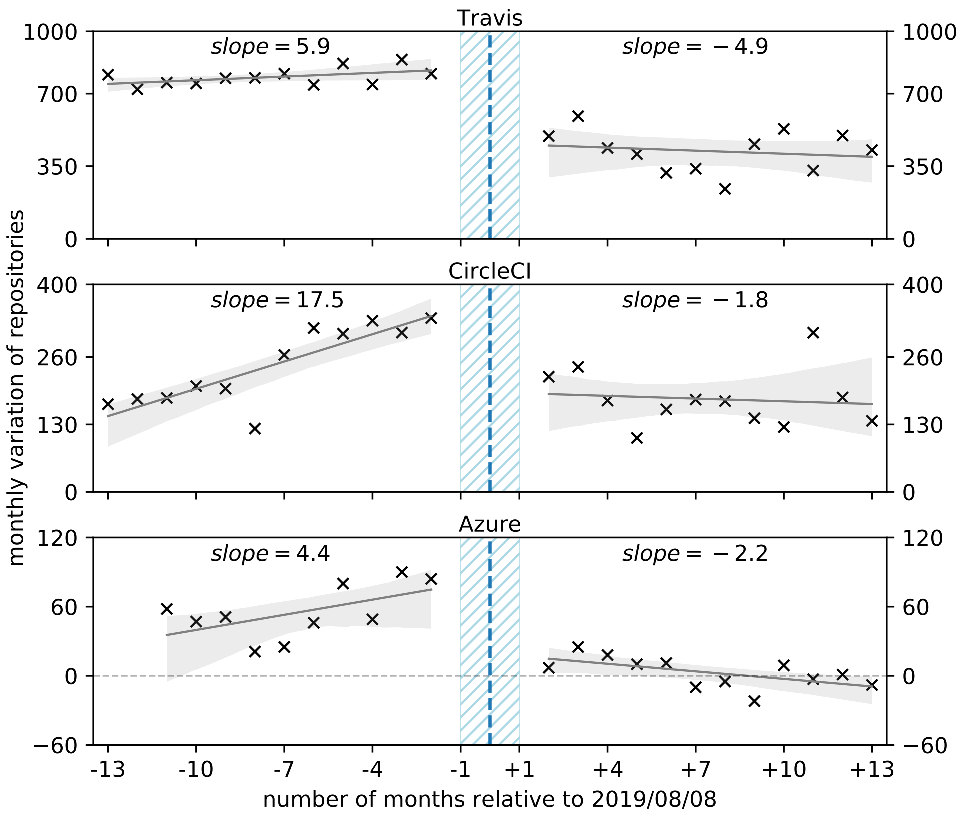


Figure 3.8: RDD analysis on the monthly variation of repositories making use of each CI (= adoptions - discontinuations) before and after introduction of GHA.

efforts. Thus, by examining the prevalence of bots, we gain insights into the extent of automation and its impact on the software engineering landscape. In Section 3.2, we delve into an analysis of the prevalence of bots, shedding light on their significance for software development automation practices.

3.2. On the prevalence of bots in GitHub

Distributed software development is, by definition, a collaborative effort involving many different persons, teams, organizations and companies. This highly collaborative software development process has led to the creation and widespread use of distributed versioning systems such as git, social coding platforms such as GitHub and GitLab, issue tracking tools such as Bugzilla, code reviewing tools such as Gerrit, and a plethora of CI services.

As witnessed by initiatives such as the CHAOSS Linux Foundation Project (https://chaoss.community) and associated software development analytics tools such as GrimoireLab (https://chaoss.github.io/grimoirelab/), it is important to assess the health of software communities by considering the activity of each contributor. Such information is also highly relevant to credit and recognise project contributors based on their activity (Hann et al., 2002), and to allow employers to identify appropriate new team members (Hauff & Gousios, 2015).

An important challenge in doing so is the presence of development robots that automate repetitive tasks to help software project contributors in their day-to-day activities. Not properly taking into account these bots may lead to incorrect or misleading conclusions, especially if such bots belong to the top contributors in software projects. This is likely to be the case, since bots are increasingly used to automate a wide range of activities, such as welcoming newcomers, reporting test coverage, updating dependencies, detecting vulnerabilities, supporting code review, submitting pull requests, verifying licensing issues, and so on (Wessel et al., 2018a).

In the following sections, we provide evidence that bots are regularly among the most active contributors in popular GitHub projects, even though GitHub does not explicitly indicate these contributors as being bots. This can be problematic for tools that aim to credit human project contributors for their activity.

3.2.1 Acknowledging contributions in collaborative development

Being able to accurately assess the contributions of project participants is valuable for many purposes. Software engineering researchers involved in empirical analyses of socio-technical activity and productivity in software projects need such data in order to understand and improve the development processes (Liao et al., 2020). Prospective employers may want to analyze developer activity profiles in order to identify skilled developers that match their job openings as closely as possible (Hauff & Gousios, 2015). Individual contributors may desire to get proper credit and visibility for their –often significant– contributions in the software projects they are involved in. They may want to use this recognition for career promotion or even to get some kind of financial support for the –often voluntary– work they spend on a project (Hann et al., 2002).

The way in which recognition is credited can differ a lot depending on the considered community. For example, OpenStack recognizes unsung heroes by discerning community contributor awards. GitHub has a similar GitHub Stars program. Initiatives such as GitHub Sponsors allow companies to sponsor open-source projects in order to help the project contributors get the recognition they deserve. Tools such as *SourceCred*[2] aim to support communities in measuring and rewarding value creation.

It is challenging to correctly determine the contributions of each project member (Kalliamvakou et al., 2009). A first challenge concerns which types of contributions should be considered (Cheng & Guo, 2019; Cánovas Izquierdo & Cabot, 2021). Typically, automated tools for identifying contributions (such as *octohattrack*[3] or *auto add contributors*[4]) provide only an impartial picture, as they tend to focus only on the types of activity that are discernible from the social coding platform (e.g., commits, pull requests, or code reviews). Other types of important contributions (e.g., finance, infrastructure, community management) are therefore often ignored (Young et al., 2021). A second challenge concerns how to identify contributors. If the same contributor uses multiple distinct accounts, or if the same account is shared by multiple contributors, identity merging and matching techniques are needed (Goeminne & Mens, 2013). Another challenge concerns how to measure activity. The real effort of contributors can only be approximated. For example, the number and size of code commits could be used as a proxy of the coding effort, but does not reflect the time required to produce such a commit, since this may depend on many external factors. Moreover, such a proxy is unable to distinguish between manual or automated activity.

Last but not least, contributors may, and regularly do, use (some of) their social coding accounts to allow automated tools (i.e., bots) to carry out repetitive activity on their behalf. Whether this is intentional or not, such usage of bots that carry out tasks on behalf of their owner can disrupt the aforementioned accreditation and recognition need. Indeed, it would be unfair to give the same recognition to a contributor whose contributions are primarily due to a bot that is committing on his/her behalf, as compared to a contributor that has invested a similar effort manually. On the other hand, there is nothing wrong with contributors that try to increase their productivity by automating some of their repetitive tasks, as long as this is not intentionally done to artificially inflate one's activity. Whether and how to give proper recognition to project contributors remains an open and difficult question.

[2]https://sourcecred.io

[3]`https://github.com/LABHR/octohatrack`

[4]`https://github.com/marketplace/actions/auto-add-contributors`

3.2.2 Distinguishing bots from humans

A first and important step to give proper recognition to project contributors consists of distinguishing human activity from bot activity. GitHub allows project contributors to discern whether certain types of activity are automated, specifically for GitHub Apps and GitHub Actions. According to the GitHub terms of service, bots are not permitted to register new GitHub accounts. However, things get more complex, since humans are permitted to set up *machine accounts* to perform automated tasks (such as a continuous integration bot), provided that a human owning the account accepts the responsibility for its actions. The problem is that the GitHub Application Programming Interface (API) does not allow to distinguish all such machine accounts from ordinary user accounts corresponding to real human activity. As a consequence, tools that want to benefit from distinguishing human users from machine users (i.e., bots) have a hard time doing so. For example, among the available tools to accredit and acknowledge contributors, *SourceCred* and *contributors-list*[5] are limited in separating human and bot contributors by relying on the GitHub API and on a user-defined list of machine accounts to do so.

This is where bot identification tools could come to the rescue. Such tools aim to distinguish bots from humans in GitHub accounts on the basis of their behaviour. The way of doing so can be quite diverse: it can be based on differences in the commenting patterns made by bots (Chapter 4), on naming conventions, or on commit activity patterns (Dey et al., 2020a). Examples of such tools are *BoDeGHa*[6] (Chapter 5) that relies on comments made in pull requests and issues, and *BoDeGiC*[7] (Chapter 6) that relies on git commit messages.

Using bot identification tools makes it easier to dissociate bot accounts from human accounts, but can still lead to incorrect detections, notably when accounts are involved in a mix of manual human activity and automated machine-generated activity (Cassee et al., 2021). Although there is still room for improving bot identification tools (Chapter 6), they can already be very helpful in identifying bots, especially in large repositories.

3.2.3 Evidence of bot contributions i n popular software projects

In order to justify the need for properly identifying bot activity in collaborative software development projects, we provide some evidence of the presence of

[5]`https://giters.com/wow-actions/contributors-list`
[6]`https://github.com/mehdigolzadeh/BoDeGHa`
[7]`https://github.com/mehdigolzadeh/BoDeGiC`

bots among the top contributors in popular open-source projects on GitHub. We selected 10 large and active open-source projects for popular programming languages (e.g., JavaScript, Java, Python, Rust). The list notably includes: *VueJS*[8], a very popular front-end framework for JavaScript with more than 40K dependent projects on NPM; *Servo*[9], an experimental browser engine written in Rust that has more than 1K contributors and nearly 40K commits; and *Cucumber-JVM*, a Java implementation of the popular test framework that has more than 53K dependent projects on GitHub.

We relied on the GitHub API to retrieve the contributors with the highest number of commits in these 10 projects, as well as their account type (i.e., user or bot) as reported by the GitHub API on 9 November 2021. This section examines the potential impact of bots that are not explicitly identified by the GitHub API on the attribution of contributors.

Fig. 3.9 depicts the top 20 contributors to these 10 popular software projects, ranked in decreasing order of activity. Contributors that are responsible for at least 1% of all commits are highlighted. We classified the contributors into three categories: *human users*, *labeled bots* as reported by the GitHub API, and *unidentified bots* that were not reported as bots by GitHub. This classification was confirmed through a manual inspection of their activities by two authors of the paper (Golzadeh et al., 2022b).

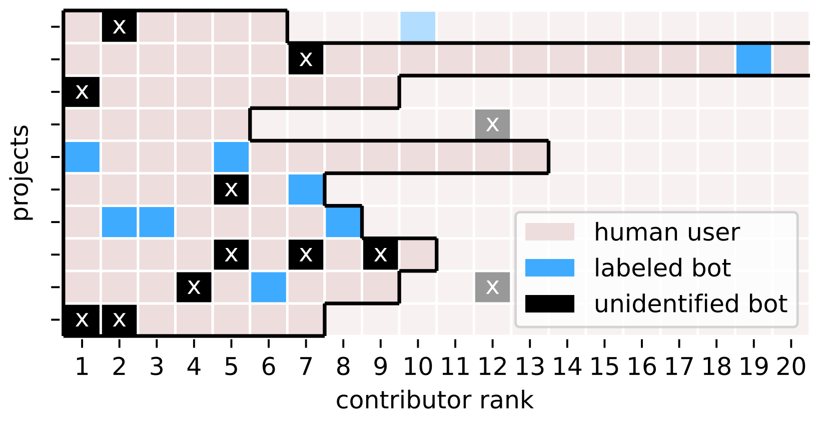


Figure 3.9: Bots observed in the top 20 most active committers in 10 popular open-source projects.

[8]`https://github.com/vuejs/vue`
[9]`https://github.com/servo/servo`

The figure shows that the considered projects have between one and three bots among the top 20 contributors. However, only less than half of the bots (9 out of 21) are reported as such by the GitHub API. The results are even more striking if we focus on the subset of contributors responsible for at least 1% of all commits: the overwhelming majority of the bots (18 out of 21) belong to those contributors and most of them (10 out of 18) are not labeled as bots by GitHub. On average, the bots are responsible for nearly one fifth of all commits in these projects.

Interestingly, we also found that some projects had explicitly credited and acknowledged bots in the list of "*people* that contributed to the project". While explicitly crediting and acknowledging contributors may encourage them to continue to contribute, the presence of bots in the contributor list may be perceived as a lack of consideration or respect towards human contributors.

3.3. Summary and conclusions

In this chapter, we explored how CI/CD systems have been used to automate development-related activities in GitHub repositories over time. We also investigated the presence of bots as another means of automation in GitHub projects. We revealed that in terms of automation many interesting observations can be made with respect to collaborative software development. Regarding continuous integration, we observed CI co-usage and migrations and witnessed how the introduction of a new competitor is changing the CI landscape. Therefore, there is a good opportunity to explore the reasons behind such CI co-usages and migrations as a future research objective.

We also showed that bots are among active contributors to projects even though they are not necessarily reported as such. This motivates the need for techniques and tools to identify bots. My thesis statement focuses on providing techniques and tools to identify bots in software development repositories. In the next chapter, we will explain how we characterize bots based on their behavior in order to develop a technique to automatically identify bot accounts.

The results of our analysis revealed that bots play an undeniable role in large collaborative software development projects. These bots seem to carry out a significant amount of work, as many of them belong to the most active project contributors. Nevertheless, many bot accounts are not labeled as such by GitHub. Understanding why they are not labeled as bots remains an open question.

Having unidentified bots among the most active contributors may be problematic. For example, the presence of such bots in a contributor list may cause difficulties when changes in the project's Contributor License Agreement (CLA) are required, since such changes require the explicit approval of all human contributors. It also becomes more difficult to give due credit to human contributors for their activities, and could even lead to bots (or rather the human owners of the associated machine accounts) receiving financial compensation for their effort.

Currently, maintainers have no choice but to manually maintain a list of active bots in their repository, by manually inspecting contributors' activities on a regular basis. While this option is feasible for smaller repositories, it is impractical to do such a manual inspection in repositories with a large number of contributors and activities. This highlights the need to rely on automatic bot identification and in turn calls for more research on accurate bot identification techniques.

Moreover, since we expect bots to become more complex and more sophisticated in the range of development activities they support and automate, there is also a need for exploiting machine learning and artificial intelligence techniques to properly detect and acknowledge the presence of bots and their specific activity patterns.

Table 3.3: Most popular CI services, number and (cumulative) proportion of repositories using them, and (cumulative) proportion of usages, in decreasing order.

			repositories			**usages**	
CI	**URL**	**first observed on**	#	%	**cum. %**	%	**cum. %**
Travis	`http://travis-ci.org`	Jun 10, 2011	53,401	58.2%	58.2%	44.9%	44.9%
GHA	`http://github.com/features/actions`	Jan 23, 2019	46,416	50.6%	90.9%	39.0%	83.9%
CircleCI	`http://circleci.com`	Jan 15, 2014	11,431	12.4%	98.1%	9.6%	93.5%
AppVeyor	`http://www.appveyor.com`	Apr 04, 2014	3,553	3.9%	98.3%	3.0%	96.5%
Azure	`http://azure.microsoft.com`	Sep 11, 2018	1,045	1.1%	98.7%	0.9%	97.3%
GitLab CI	`http://docs.gitlab.com/ee/ci`	Sep 02, 2015	1,018	1.1%	99.1%	0.9%	98.2%
Jenkins	`http://www.jenkins.io`	Mar 30, 2016	1,008	1.1%	99.6%	0.8%	99.0%
Others	N/A	Oct 23, 2013	1,138	1.2%	100.0%	1.0%	100.0%

Table 3.4: RDD analysis of the monthly variation of CI usage, before and after official introduction of GHA.

CI	**event** (error)	**time** (error)	**time_after** (error)	**constant** (error)	$\mathbf{R^2}$
CIs landscape	146.0 (172.4)	25.8 (17.6)	**132.1***** (24.8)	**943.8***** (129.3)	0.94
Travis	**-357.1***** (64.5)	5.9 (6.6)	-10.8 (9.3)	**741.0***** (48.4)	0.86
CircleCI	**-148.8**** (41.4)	**17.5***** (4.2)	**-19.3**** (6.0)	**128.5***** (31.1)	0.57
AppVeyor	-16.5 (20.1)	-1.4 (2.1)	-0.1 (2.9)	24.2 (15.1)	0.38
Azure	**-57.8***** (13.6)	**4.4*** (1.8)	**-6.6**** (2.2)	22.2 (14.1)	0.79
GitLab CI	-0.7 (4.6)	-0.4 (0.5)	-0.4 (0.7)	**23.1***** (3.5)	0.46
Jenkins	17.6 (10.8)	-2.2 (1.1)	2.4 (1.6)	**22.6*** (8.1)	0.22
	*** p <0.001, ** p <0.01, * p <0.05				

CHAPTER 4

Distinguishing characteristics of bots

"The truth is always in harmony with herself, and is not at variance with herself, nor can she ever be."

Zoroasters

In Chapter 3, we showed that CIs and bots are an important part of collaborative software development and that there is a need for an automatic technique to identify bots in GitHub repositories. To validate such a technique, we need to have a ground-truth dataset of bots and humans active in collaborative development. Given that such a dataset was not available at the time we conducted the study, we had to create it ourselves. To do so, we randomly selected a large set of GitHub repositories and extracted the data of accounts active in these repositories and their corresponding PR and issue comments. In order to be able to identify the type of these accounts, we developed a rating application that allowed us to tag these accounts as humans and bots. Using this rating application, we created a dataset composed of 5,000 accounts, including 527 bot accounts and 4,473 human accounts.

The dataset allows us to study the behavior of bot accounts and human accounts. The goal is to identify bots by analysing their behavior and activities in a way that helps us recognize them. As an example, we show that bot accounts tend to communicate with other accounts using very similar comments or a small set of similar comments, and bots are less likely to submit empty comments since they are expected to generate informative messages. We characterize bots based on such distinctive behaviors.

4.1. Terminology

In the context of this chapter, we will consistently use the following terminology. We use the term ***bot*** to refer to a GitHub bot, defined by Wessel (Wessel et al., 2018b) as "*a task-oriented bot, responsible for automating well-defined tasks on GitHub repositories. A GitHub bot behaves like a human user, serving as an interface between users and services.*"

Since our study focuses on distributed software development on GitHub, we use the term ***repository*** to refer to a GitHub repository. Contributors to a repository can be identified by their unique (GitHub) ***account***. Contributions to a repository can take different forms, such as code ***commits***, ***issues*** and ***pull requests (PR)***. The focus of this chapter will be on issues and PRs.

Contributors can add (uniquely identifiable) ***comments*** to PRs and issues in a repository. We use the term ***commenter*** to refer to the GitHub account having provided this comment. We also use the term comment to refer to its actual textual content. Since a commenter can be either a bot or a human contributor, we will refer to them as ***bot commenter*** and ***human commenter***, which we will abbreviate to ***bot*** and ***human***, respectively.

4.2. Data extraction

In order to be able to evaluate an automated algorithm to detect bots based on their commenting activity in GitHub issues and pull requests, a ground truth dataset is required. Such a ground truth dataset indicates, given a contributor commenting in an issue or a pull request, whether this contributor is a human or a bot. To be effective and representative, the ground truth dataset should be large enough, i.e., it should cover a considerable number of GitHub repositories, contributors, issues and pull requests.

Since we did not encounter any such representative ground truth dataset in the research literature, we set out to create it ourselves. To do so, we downloaded and manually examined comments from thousands of issues and pull requests, labelling each contributor either as a bot or a human commenter. Despite the considerable effort needed to create such a dataset, it was a worthwhile endeavour, since it will be a valuable resource for other researchers as well.

This section explains how we proceeded to create and validate our ground truth dataset, from the raw data we downloaded to the process of rating and labelling each contributor.

A common starting point to download data from GitHub is GitHub public event endpoint[1] which allows researchers to access the most recent activities on the entire GitHub repositories. Our goal is to identify bot and human commenters based on the comments they made in issues and pull requests of collaborative software development repositories on GitHub.

GitHub is one of the leading online collaborative development platform. As of November 2020, GitHub reported having over 48 million users and more than 195 million repositories (including at least 37 million public repositories).

Following the guidelines provided by Kalliamvakou et al. (2014b), we want to avoid repositories that have been created merely for experimental or personal reasons, or that only show sporadic traces of issue and PR comments. Moreover, since our focus is on software development repositories, we want to exclude repositories that are not related to software development. To comply with these constraints, we relied on libraries.io (Katz, 2020), a monitoring service indexing information for several million packages being distributed through 37 software package registries, such as npm, PyPI, etc.

We downloaded the data dump of January 2020[2] containing, among others, links to the GitHub repositories related to these distributed software packages. Since it contains more than 3.3 million GitHub repositories, we randomly selected around 136K of them as the starting point of our dataset creation process. For each of these repositories, we extracted on 16 February 2020 the last 100 comments of the last 100 issues and pull requests using GitHub's GraphQL API. This resulted in over 10 million comments covering a period of more than 10 years (ranging from 17 December 2009 to 15 February 2020). These comments were made by more than 837K distinct contributors, corresponding to more than 3.5 million issues and pull requests. The extracted comments also include the textual description of each considered PR. While the GitHub API does not consider PR descriptions as comments, we do, since the GitHub web interface does not visually distinguish them from other comments.

Since our goal is to distinguish between bots and human contributors based on their comments, we require a sufficiently large number of comments for each commenter. Hence, we decided to exclude commenters who made fewer than 10 comments based on a threshold we identified in one of our studies (Golzadeh et al., 2020). At this stage of the process, the dataset contains 6,307,489 comments belonging to 79,342 contributors, spanning 42,492 repositories.

Since this is too much data to process manually, we extracted a subset covering 5,082 commenters. This subset was composed of 4,644 randomly selected commenters to which we manually added 438 extra commenters that

[1] `https://docs.github.com/en/graphql`

[2] Version 1.6 on `http://doi.org/10.5281/zenodo.3626071`

Table 4.1: Summary of the dataset characteristics.

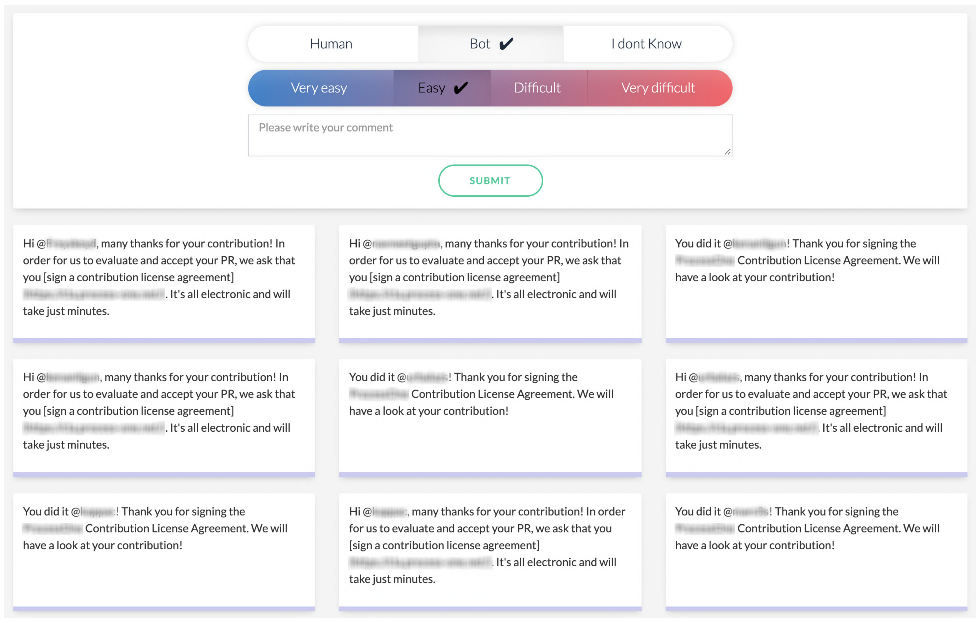
raw dataset	**number**
GitHub repositories	136,529
↪ from # distinct owners	84,983
issues	1,588,363
pull requests (PR)	1,951,705
issue and PR comments	10,874,611
↪ from # distinct commenters	873,489
selected subset	
GitHub repositories	3,975
↪ from # distinct owners	3,425
issues	50,241
pull requests (PR)	136,750
issue and PR comments	301,557
↪ from # distinct commenters	5,082

are more likely to correspond to bots based on previous studies (Golzadeh et al., 2020; Wessel et al., 2018b) (52 cases), or because they contained a specific substring in their GitHub account name (386 cases). The substrings we considered were "bot", "ci", "cla", "auto", "logic", "code", "io" and "assist". By doing so, we increased the likelihood of having a sufficient number of bots in the dataset.

The resulting subset contains 5,082 commenters and covers 3,975 repositories, 186,991 issues and pull requests, and contains 301,557 comments. Table 4.1 summarizes the main characteristics of the considered datasets.

4.3. Data labelling and rating process

The next step to create a ground truth dataset is to manually identify bots and humans. To ease this process, we developed a web application through which the list of comments of each commenter was presented to at least two raters among the four authors of the paper (Golzadeh et al., 2021). Comments were displayed by batches of 20, starting with the most recent comments first, and the rater had an option to display more comments if needed. The account name of the commenter was not revealed to avoid bias, as the goal was to classify



Human [✓ Bot] I dont Know

Very easy [✓ Easy] Difficult Very difficult

Hi @[redacted], many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement][redacted]. It's all electronic and will take just minutes.

Hi @[redacted], many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement][redacted]. It's all electronic and will take just minutes.

You did it @[redacted]! Thank you for signing the [redacted] Contribution License Agreement. We will have a look at your contribution!

Hi @[redacted], many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement][redacted]. It's all electronic and will take just minutes.

You did it @[redacted]! Thank you for signing the [redacted] Contribution License Agreement. We will have a look at your contribution!

Hi @[redacted], many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement][redacted]. It's all electronic and will take just minutes.

You did it @[redacted]! Thank you for signing the [redacted] Contribution License Agreement. We will have a look at your contribution!

Hi @[redacted], many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement][redacted]. It's all electronic and will take just minutes.

You did it @[redacted]! Thank you for signing the [redacted] Contribution License Agreement. We will have a look at your contribution!

Figure 4.1: Anonymised screenshot of the rating application in action.

commenters based on their comments only. The rater could select whether the commenter is considered as a “Bot” or a “Human”. In case a rater was uncertain whether the commenter was a bot of a human being, a third option could be selected: “I don’t know”. Furthermore, the rater was asked to select a difficulty level among “Very easy”, “Easy”, “Difficult” and “Very difficult” for his decision.

Fig. 4.1 shows a screenshot of the rating application in action. For the specific example being shown, raters could easily decide that the commenter is a bot based on the content and repetitiveness of all visible comments.

In total, 5,082 commenters were rated, ending up with exactly 5,000 commenters after having filtered out 82 commenters during the following process. We considered this amount as more than sufficient for our empirical study, especially because of the very time-consuming manual effort that was involved in the rating process. The rating process was performed in two steps to come with an optimal inter-rater agreement, relying on Landis agreement levels (Landis & Koch, 1977). The rating process is summarized in Fig. 4.2. Each commenter was initially rated by two distinct raters. All cases that were agreed either as bot or human were included in the ground-truth dataset. In order to assess the

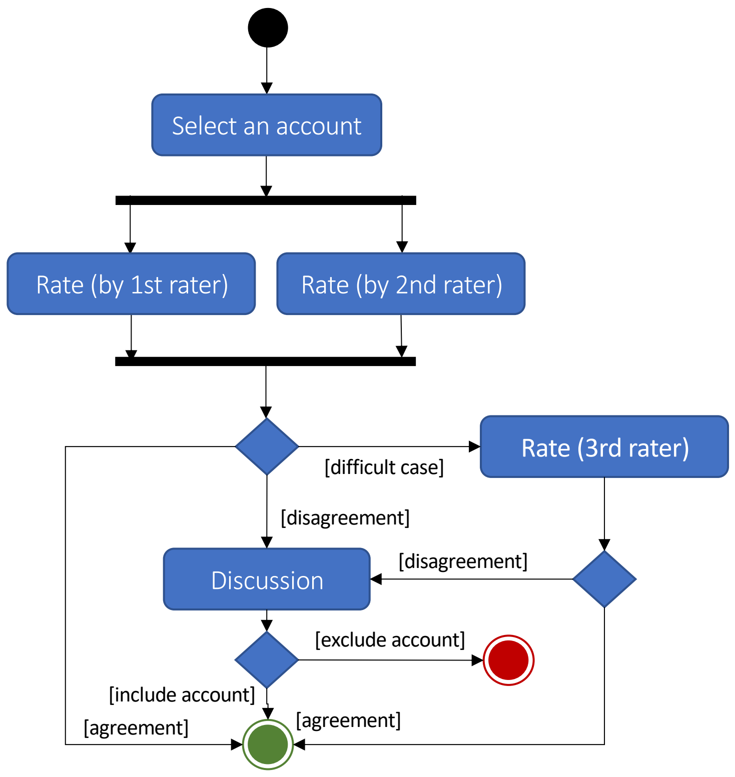


Figure 4.2: Workflow of the rating process.

reliability of the ground-truth dataset, we computed the inter-rater reliability (IRR) (Campbell et al., 2013) between each pair of ratings based on Cohen's kappa κ (McHugh, 2012). The results are presented in Table 4.2.

The first step of the rating process ended up with 472 bots and 4,364 humans, with a "*substantial*" agreement ($\kappa = 0.84$) between raters. At the end of this step, there were 246 cases because they were either not agreed (177 cases) or agreed as "I don't know" (69 cases). Additionally, 91 cases evaluated as "difficult" or "very difficult", leading to a total of 268 cases for the second step.

In a second step, a third rater was involved for the cases that were identified as "difficult" or "very difficult" during the first step. All raters then discussed together all cases for which an agreement could not be achieved, or the cases where the third rater disagreed with one of the two former ones. During these

Table 4.2: Summary of two-step rating process.

	first step	**second step**
commenters agreed as *bot*	472	527
↪ from # repositories	457	505
commenters agreed as *human*	4,364	4,473
↪ from # repositories	3,425	3,515
proportion of bots	9.8%	10.5%
commenters agreed as "I don't know"	69	–
commenters without agreement	177	4
commenters agreed as "mixed"	–	78
κ agreement score	0.84	0.96

discussions, the raters sometimes relied on additional information (e.g., they looked at the GitHub account of the commenter, at time intervals between comments, the overall activity of the account, etc.) to come to a decision.

The large majority of discussed cases were resolved on the basis of an unanimous decision between raters, leading to a very high inter-rater reliability ($\kappa = 0.96$). At the end of the second step, only 82 cases were left out of the ground-truth dataset, either because no agreement could be reached (4 cases), or because the raters agreed on the "mixed" nature of these commenters. These "mixed" commenters correspond to human commenters that relied on automatic tools to generate comments, therefore "mixing" the behaviour of a human and a bot at the same time.

For example, some of these accounts rely on an automated tool to facilitate code review by sending PRs to *Reviewer*, a code review tool for GitHub. Other examples include the use of tools such as *StyleCI* to improve code style, or *semantic-release* to automatically determine the next version number of a release, generate release notes and publish a package. We will discuss these "mixed" commenters in more details in Section 5.5.

This left us with 5,000 commenters, of which 527 (i.e., 10.5%) are bots. Table 4.3 summarizes the characteristics of final ground-truth dataset. Since we believe such a ground-truth dataset is valuable for the research community (e.g., to have a list of known bots, to study their characteristics or to train other models), we shared it publicly on `http://doi.org/10.5281/zenodo.4000388`. This dataset contains the name of the repository, the name of the commenter and whether it is a bot or a human. Due to GDPR regulations and in order to

Table 4.3: Summary characteristics of final ground truth dataset.

number of...	**bot**	**human**	**total**
commenters	527	4,473	5,000
repositories with at least 1 commenter	505	3,515	3,909
comments	28,287	268,504	296,791
issues with at least 1 commenter	2,749	46,959	49,623
PRs with at least 1 commenter	16,937	118,896	134,208

protect GitHub users' privacy, we did not provide additional information (e.g., their comments).

4.4. Characteristics of bots

In this section, we explain the features that will be used by the classification model to distinguish bots from human commenters. These features include the number of comment patterns, the number of (empty) comments, and the number of comments within each pattern. The following subsections explain these features and the rationale behind their selection.

4.4.1 Text distance between comments

Based on the assumption that bots perform more repetitive and automated tasks, we hypothesise that bot commenters exhibit more repetitive comments than human commenters. Consequently, we expect comments belonging to a bot to exhibit more similarity than comments belonging to a human commenter. In order to measure the similarity between comments of each commenter, both in terms of content and structure, we rely on text distance metrics that are commonly used for this purpose in natural language processing. The two metrics we consider are the Jaccard (Jaccard, 1912) and Levenshtein (Levenshtein, 1966) distances. The first one aims to quantify the similarity of two texts based on its content, and the second one captures the structural difference by counting single character edits.

More precisely, the Jaccard distance $J(C_1, C_2)$ measures the distance between two texts C_1 and C_2 by comparing the number of distinct common words in C_1 and C_2 with the total number of distinct words in C_1 and C_2. If

$words(C)$ denotes the set of words in C, then $J(C_1, C_2)$ is computed as:

$$\mathcal{J}(C_1, C_2) = 1 - \frac{|\ words(C_1) \cap words(C_2)\ |}{|\ words(C_1) \cup words(C_2)\ |}$$

The second distance we consider is the Levenshtein edit distance $lev(C_1, C_2)$ that measures the difference between two character sequences C_1 and C_2 by counting the minimum number of single-character edits (insertion, deletion, or substitution) required to convert C_1 into C_2. We rely on its normalized version, computed as:

$$\mathcal{L}(C_1, C_2) = \frac{lev(C_1, C_2)}{max(|C_1|, |C_2|)}$$

To support our assumption that comments made by a bot have higher similarity than comments made by a human, we computed for each commenter in the ground truth dataset the Jaccard and Levenshtein distances between all pairs of comments belonging to that commenter. In order to compute the Jaccard distance, we first needed to split comments into words, a process also known as *tokenization*. To do so, we relied on spaCy, an "industrial-strength natural language processing library"[3] that notably offers a fast but robust tokenization algorithm, among others.

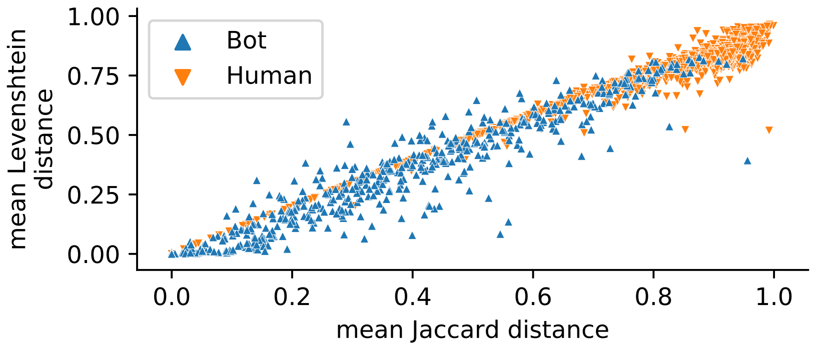


Figure 4.3: Mean Levenshtein and Jaccard distances between pairs of comments, per commenter.

Fig. 4.3 shows the mean Levenshtein and Jaccard distances for each commenter, distinguishing between bots (blue triangles) and humans (orange triangles).

[3]`https://spacy.io`

We observe that many humans are grouped in the top right part of the figure, i.e., they have high mean values for both distances. On the other hand, most bots have lower values for their mean distances. For instance, 91.6% of all bots have mean Jaccard and Levenshtein distances below 0.75. For comparison, only 7.2% of all human commenters exhibit mean Jaccard and Levenshtein distances below 0.75.

Despite this, there is still a lot of overlap between bots and humans in Fig. 4.3, indicating that the mean distances are not enough to properly distinguish between bots from humans. By manually inspecting the comments belonging to bots having high mean distances, we found that their comments usually form sets of similar comments. Even if the distance between comments in a set (i.e., intra-set distance) is low, the distance between comments belonging to different sets (i.e., inter-set distance) is high. As a consequence, the overall mean distances between all comments tends to remain high, rivalling the distances observed for most human commenters.

We found many of these cases. One example is the bot that was identified in Fig. 4.1. We observe that it has two different sets of similar comments. The first set consists of comments of the form "*You did it @...! Thank you for signing the ... Contribution License Agreement. We will have a look at your contribution!*". The second set consists of comments of the form "*Hi @..., many thanks for your contribution! In order for us to evaluate and accept your PR, we ask that you [sign a contribution license agreement] ... It's all electronic and will take just minutes.*". The mean distance between pairs of all 20 comments belonging to the first set (i.e., intra-set distance) is very low (0.06 and 0.08 for Levenshtein and Jaccard distance respectively) and even lower (0.04 and 0.05 respectively) for the second set of 27 comments. However, the intra-set distance (i.e., the distance obtained by comparing comments from the first pattern with comments for the second pattern) is much much higher (0.70 and 0.81 for Levenshtein and Jaccard distance respectively). Consequently, the overall mean distances between all pairs of comments are 0.37 for Levenshtein and 0.43 for Jaccard distance. These distances are usually observed for human commenters, not for bots.

After scrutinizing the comments, we came to the conclusion that bots follow some patterns in their comments. Assume that all comments produced by a bot belong to a limited set of comment patterns. Within each cluster of comments corresponding to the same comment pattern, the mean distance will be low. Yet, when considering all comments together, the mean distance can remain high, since the distance between comments belonging to different clusters can be high. Hence, to obtain a better way to distinguish bots from humans, we decided to compute the similarity between sets of comments instead of all comments for each commenter.

4.4.2 Repetitive comment patterns

Since high mean distances between comments of a commenter could correspond to either a human or, in many cases, to a bot having sets of similar comments, we cannot exclusively rely on these mean distances to distinguish between bots and humans. However, we observed that bots tend to have sets of many similar comments (i.e., they follow comment patterns), while we found that most comments from humans are unique and only a few of them seem to follow a pattern (e.g., "*Thank you!*", "*LGTM*"[4] or "*+1*"[5]). Based on this observation, we expect bots to have a lower number of comment patterns than humans. In order to capture these comment patterns, we rely on a clustering algorithm. Clustering aims to group items into sets ("clusters"), in such a way that items belonging to the same cluster are more similar than items belonging to different clusters.

We selected DBSCAN (Density Based Spatial Clustering of Applications with Noise) (Ester et al., 1996), a well-known density-based clustering algorithm that notably has the ability (i) to generate clusters of unequal size (i.e., we can have patterns with unequal numbers of comments), (ii) to generate a single cluster if needed (e.g., a commenter whose comments are all the same), and (iii) to generate single item clusters (e.g., a commenter whose comments are all very different). Additionally, DBSCAN permits not to specify the number of clusters in advance, fitting our use case wherein we do not know the number of patterns of each commenter in advance.

Since we aim to capture both the structural and content distance between comments, we rely on a combination of the Levenshtein and Jaccard distance, defined as follows:

$$\mathcal{D}(c_1, c_2) = \frac{\mathcal{L}(c_1, c_2) + \mathcal{J}(c_1, c_2)}{2}$$

For each commenter, we computed $\mathcal{D}(c_i, c_j)$ for each pair (c_i, c_j) of comments. The resulting distance matrix, one per commenter, is then passed to DBSCAN to group the comments based on their similarity. Fig. 4.4 reports on the number of patterns (i.e., clusters), distinguishing between bot and hu-

[4]Shorthand for "Looks Good To Me", a common way among GitHub users to agree with what is proposed in a pull request.

[5]This is another common way of expressing agreement with what was proposed in the previous comment or in the issue or PR description.

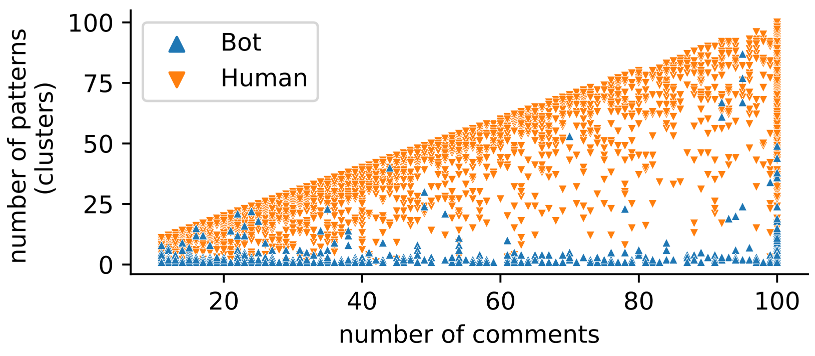


Figure 4.4: Number of comment patterns (clusters) and number of considered comments per commenter.

man commenters. Since the number of patterns could depend on the number of comments, we report on the number of patterns relative to the number of considered comments.

Compared to Fig. 4.3 we can observe a much clearer separation between bots and humans based on the number of comment patterns and the number of comments, although it is not perfect. We observe that most humans are along the diagonal line which indicates that the number of patterns is close to the number of comments, and that almost all bots are along the horizontal axis. This means that the number of comment patterns for bots remains stable, and low, regardless of the number of comments they made. This confirms our assumption that bots have a limited set of comment patterns, contrarily to humans that seems to make much more varied comments.

4.4.3 Inequality between comments in patterns

Although we expected human comments to be mostly non-repetitive (i.e., each comment corresponds to a different pattern), we found instances in which a human commenter had a non-negligible number of repetitive comments (e.g., "*Thank you!*", "*LGTM*" or "*+1*") alongside other messages. This leads to having human commenters whose number of comment patterns is much lower than the number of comments, which is exactly the assumption we had for bots due to their repetitive comments. However, we found that those human commenters correspond to cases having at the same time a few patterns with many comments and many patterns with a few (mostly single) comments. On the other hand, bots exhibit single comment patterns less often. For instance, among the 2,431 patterns corresponding to bots, 50% are composed of a single comment, while this proportion is much higher (95.9%) for the 230,711 patterns we have for humans.

This observation lead us to consider the inequality in the number of comments in each pattern as a supplementary feature to distinguish between bots and humans. The Gini coefficient (Dorfman, 1979) provides a way to quantify the inequality (i.e., the distribution) of the number of comments for each pattern. A value of 0 expresses perfect equality (i.e., each comment pattern consists of the same number of comments). A value of 1 expresses maximal inequality among values (i.e., a few patterns capture many comments, and the remaining comments are spread into many single-comment patterns).

Let us consider the example of a specific human commenter in our dataset. This human made 73 comments belonging to 12 patterns. 9 of these patterns have exactly one comment. The other ones correspond to "*LGTM*" (37 comments), "*##Fixes{Number}*" (22 comments) and "*lgtm*" (5 comments). As a result, the Gini coefficient for this commenter is very low 0.04, since most patterns (9 out 12) have the same number of comments. Let us compare this to a bot in our dataset with a similar number of comments (61) and comment patterns (10). The number of comments in each pattern is more unequally distributed, ranging from 1 to 49 comments per pattern, a consequence of much more repetitive messages. As a result, its Gini coefficient is much higher, namely 0.52.

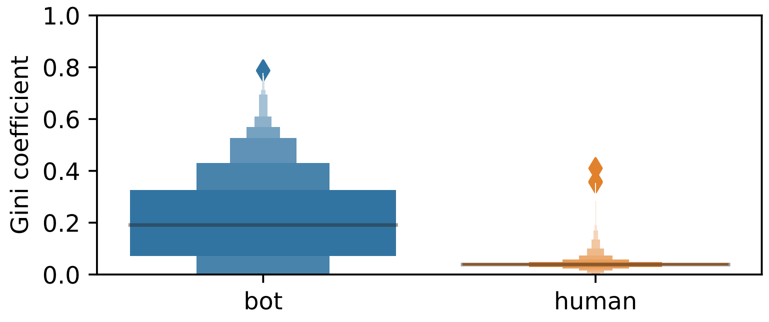


Figure 4.5: Distribution of Gini coefficient for bot and human commenters.

Fig. 4.5 shows the distribution of the Gini coefficient for all bots and humans in our dataset, by means of boxen plots (Hofmann et al., 2011). We observe that humans exhibit a lower inequality than bots with respect to the spread of comments within patterns. We statistically compared these distributions using a Mann-Whitney-U test (Mann & Whitney, 1947). The null hypothesis, stating that the two distributions are the same was rejected ($p < 0.001$), indicating a statistically significant difference between the two distributions. The effect size turned out to be *large* (Cliff's delta $|d| = 0.58$) (Cliff, 1993; Romano et al., 2006). This confirms that humans tend to have a lower inequality than bots, a consequence of many of their patterns containing a single comment. Therefore, the Gini coefficient can help in distinguishing between bots and humans.

4.4.4 Number of comments

In addition to the number of patterns and the unequal distribution of comments within patterns, we also consider the number of comments made by each commenter as a feature for our model. This feature makes it possible to distinguish between commenters having a similar number of patterns. Indeed, consider for example two commenters having exactly 10 patterns. Assume they have respectively 10 and 100 comments. The first commenter is likely to be a human (since it has 10 patterns each containing exactly one comment, i.e., all comments are different), while the second one is more likely to be a bot.

4.4.5 Number of empty comments

We also consider the number of empty comments as a feature for our model. Indeed, during the rating process we found that a non-negligible proportion (6.5%) of the considered comments were empty. The presence of such comments in the dataset may seem strange. Even if the GitHub user interface does not allow empty comments in a discussion, it does not prevent comments to be composed of white characters. Moreover, the GitHub user interface allows the creation of pull requests whose description is empty. Since this description is the very first comment of a pull request, it explains why we found empty comments in the dataset.

Interestingly, we found that empty comments are mostly created by human commenters and not by bots. For instance, only 7% of all bots generated at least one such comment, whereas this proportion reaches 41.2% for human commenters. This should not come as a surprise, since one could expect bots mainly to generate informative comments and, by definition, empty comments are uninformative. Consequently, we decided to consider the number of empty comments as a feature of our classification model.

4.5. Summary and conclusions

This chapter examined the commenting activity of bots in GitHub issue and pull requests, and identified four main features that characterize their behavior. Based on the analysis in Section 4.4 we decided to use four distinct features for commenters to train the classification model: (i) the number of comment patterns; (ii) the inequality between comments in patterns; (iii) the total number of comments for the commenter; and (iv) the number of empty comments. Compared to human accounts, (i) bots were found to have fewer comment patterns, (ii) they tend to make more informed comments, so they rarely leave empty comments, (iii) while bots have diverse numbers of comments in each pattern, humans mostly have comment patterns with single item, and (iv) the number of comments is linked to the number of comment patterns. These parameters can be used to distinguish bots from human accounts.

In order to detect bots in social coding platforms it is important to characterize bots based on their behaviour. Yet, such characterization is not sufficient for an automatic method of identifying bots. The development of an automatic technique to detect bots based on such characteristics requires the development of a classification model. To train and evaluate a classification model in the next chapter, we will use the ground truth dataset we prepared in this chapter.

CHAPTER 5

A classification model for bot detection

"Science is the most reliable guide in life."

Avicenna

Chapter 4 identified features that can help to distinguish bots from human accounts. We provided visual and statistical evidence of the distinction between humans and bots but this is not sufficient since we need an automatic method to identify bots. Thus in this chapter we implement a classification model to identify bots on top of these features.

We employed machine learning algorithms in order to classify accounts based on the numerical features that we identified in the previous chapter. The output of the classifier should be either "bot" or "human", therefore we need a binary classification technique. Among the classifiers having the ability to perform binary classification, we evaluated 5 different well-known classifiers, we follow a grid-search cross-validation to evaluate these classifiers with different parameters in order to select the best classifier. We evaluate the accuracy of the classification model based on a test set of unseen data and we provide the model accuracy based on common machine learning accuracy metrics. In order to make the classification model more usable for practitioners and researchers, we implemented a command-line tool based on it. We present the structure and how BoDeGHa (Bot Detector for GitHub accounts) works in classifying accounts of a given GitHub repository.

5.1. A classification model to identify bot accounts

A wide variety of algorithms can be used to construct a classification model. In this section we compare different classification algorithms to determine which one is the most appropriate to distinguish between bot and human commenters. Machine learning algorithms fall under three categories: supervised learning, unsupervised learning, and reinforcement learning (Mitchell, 1997). In this work, we compare different supervised learning algorithms to determine which algorithm is the most appropriate. We do not use unsupervised models because we have classes of commenters and we want to predict the labels based on historical data. We do not use reinforcement learning either as this technique focuses on learning as events happen, whereas we investigate learning from historical data. Among the classifiers having the ability to perform binary classification, we consider Decision Trees (DT) (Safavian & Landgrebe, 1991), Random Forest (RF) (Breiman, 2001; Frank & Hall, 2001), Support Vector Machines (SVM) (Gunn, 1998), Logistic Regression (LR) (Burridge, 1991), and k-Nearest Neighbours (kNN) (Aha et al., 1991). Since the performance of these classifiers could depend on the input parameters, we follow a standard workflow of hyper-parameter tuning using a grid-search cross-validation process (Witten et al., 2011) (see Fig. 5.1). To do so, we rely on scikit-learn (Pedregosa et al., 2011), a well-known machine learning library for Python.

We first divided the ground-truth dataset into two disjoint sets: a training set containing 60% of the data that will be used in a grid-search cross-validation process to determine the best input parameters and the best classifier, and a test set composed of the remaining 40% that will be used to evaluate the performance of the selected classifier and parameters on new data. Since we have many more humans than bots in our datasets, we relied on a stratified train-test split method to create these two sets with the same ratio of bots and humans.

Selecting an appropriate model with the best possible parameters requires hyper-parameter tuning. Based on the supported parameters of each classifier, we implemented a grid-search process based on a limited set of values for each parameter. For example, DT and RF were evaluated by setting the split criterion to *Gini* and *entropy*, among others. Doing so resulted in 91 different classifiers. To address the class imbalance problem (He & Garcia, 2009) and avoid affecting the performance of the classifiers (Grbac et al., 2013), we rely on a cost-sensitive learning approach (Elkan, 2001). Practically, this means we set the *class weight* parameter in scikit-learn to *balanced* for each supported

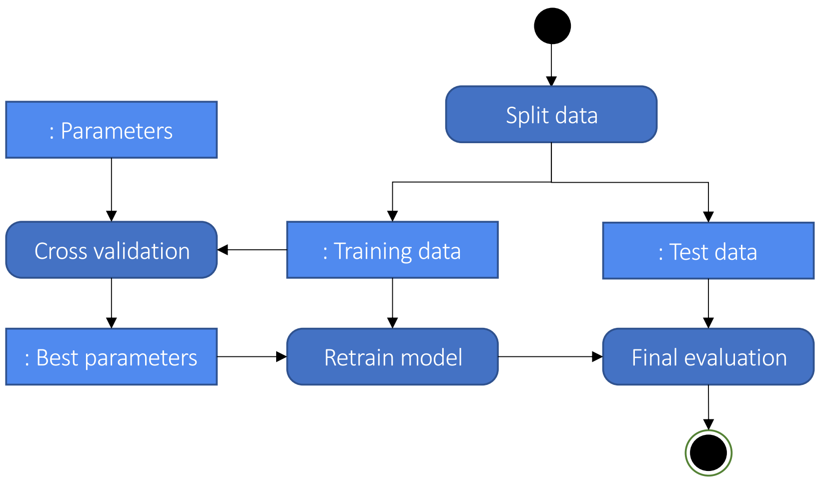


Figure 5.1: Standard workflow for grid-search cross-validation

classifier.

We then trained and evaluated the performance of all classifiers using a 10-fold cross-validation process. This approach splits the dataset into 10 subsets of equal size, and for each fold a model is trained using 9 subsets and is evaluated on the remaining one. The overall performance of the model is averaged from the performance of these 10 models. To ensure that the created subsets preserve the same proportion of bots and humans as in the complete training set, we relied on a *stratified shuffle split* to create them.

The performance of the resulting models is measured using the classical metrics of precision P, recall R and $F1$-score. We use these metrics for the population of each class (i.e., for bots B and humans H). For the entire population we computed the *weighted* version of these metrics to take into account the class imbalance. We aim to achieve an as high $F1$-score as possible. Since our goal is to identify bots, we also strive to keep bot recall $R(B)$ high enough, given that the population of bots is significantly smaller than the population of humans, and that it is much easier and faster to recover from humans misclassified as bots than the opposite. All these metrics are summarized in Table 5.1, and are defined in terms of the number of *true positives* $\boldsymbol{TP}$ (the number of bots that are correctly classified as such by the model), *true negatives* $\boldsymbol{TN}$ (the number of humans that are correctly classified as such by the model), *false positives* $\boldsymbol{FP}$ (humans that are wrongly classified as bots), and *false negatives* $\boldsymbol{FN}$ (bots that are wrongly classified as humans).

Table 5.1: Definitions of precision, recall and $F1$-score.

population	**precision** P	**recall** R	$F1$**-score**
bots B	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{2\times P(B)\times R(B)}{P(B)+R(B)}$
humans H	$\frac{TN}{TN+FN}$	$\frac{TN}{TN+FP}$	$\frac{2\times P(H)\times R(H)}{P(H)+R(H)}$
$B\cup H$	$\frac{P(B)\times\lvert B\rvert+P(H)\times\lvert H\rvert}{\lvert B\rvert+\lvert H\rvert}$	$\frac{R(B)\times\lvert B\rvert+R(H)\times\lvert H\rvert}{\lvert B\rvert+\lvert H\rvert}$	$\frac{2\times P\times R}{P+R}$

Table 5.2: Precision, recall and $F1$-score of the best classifiers per family of classifiers (in descending order of $F1$-score).

	bots		**humans**		**overall** $(B\cup H)$		
classifier	$P(B)$	$R(B)$	$P(H)$	$R(H)$	P	R	F_1
RF	0.932	0.916	0.990	0.992	0.984	0.984	0.984
kNN	0.943	0.853	0.983	0.994	0.978	0.979	0.978
SVM	0.876	0.925	0.991	0.984	0.979	0.978	0.978
DT	0.882	0.884	0.986	0.985	0.975	0.974	0.974
LR	0.839	0.931	0.992	0.978	0.975	0.973	0.974
ZeroR	-	0.000	0.893	1.000	0.798	0.893	0.843

Following the grid-search cross-validation process described above, we trained and obtained 91 classifiers. For each of them, we computed the resulting *bot*, *human* and *overall* precision, recall and $F1$-score. Table 5.2 reports on these metrics, in descending $F1$-score order. To ease readability, rather than reporting on all 91 classifiers, we selected for each classifier category (e.g., DT, RF, ...) the instance whose parameters resulted in the highest $F1$-score. We also compared the precision, recall and $F1$-score of these classifiers against a baseline classifier, ZeroR. ZeroR is a very simple classifier that has no predictive power: it ignores the features and always predicts the majority class (i.e., "human" in our case). We observe that all classifiers exhibit a high overall performance (in terms of precision, recall and $F1$) and surpass ZeroR by a wide margin.

The overall scores for R, P and $F1$ of all classifiers are consistently higher than the ZeroR baseline, and range between 0.974 and 0.984. Even though the best SVM and LR classifiers have higher bot recall $R(B)$ than the best RF classifier (0.925 and 0.931 compared to 0.916, respectively), the overall R, P

and $F1$ scores are highest for the RF classifier. We therefore decided to use the best RF classifier, which was obtained with the *entropy* split criterion, 10 estimators (i.e., trees) and a maximum depth of 10 for these trees.

5.2. Evaluation of the model

In this subsection, we aim to evaluate the actual performance of that model on data that were not used to train the model, i.e., on new data contained in the test set. Following the workflow presented in Fig. 5.1, we start by constructing a new classification model instance based on the selected RF classifier, its parameters, and the training set containing 60% of the ground-truth dataset.

We evaluate and report the accuracy of the model based on the test set, corresponding to the remaining 40% of the ground-truth dataset. This test set includes 2,000 commenters, of which 1,789 are humans and 211 are bots. The evaluation results are reported in Table 5.3.

Table 5.3: Evaluation of the classification model using the test set.

	classified as bot	classified as human	Precision	Recall	F1
Bot	TP: 192	**FN: 19**	0.94	0.91	0.92
Human	**FP: 13**	TN: 1776	0.99	0.99	0.99
weighted avg			0.98	0.98	0.98

We see that most bots and humans are correctly classified by the model. For instance, only 19 out of 211 bots were misclassified as humans (**FN**), and only 13 out of 1789 humans were misclassified as bots (**FP**). The overall $F1$-score is very high (0.98), a consequence of the high precision (0.98) and high recall (0.98) of the model. Thanks to the fact that we have taken into account class imbalance during the training phase, these high scores can also be observed individually for each class, even if the precision and recall for bots is slightly lower than for humans. These results confirm what we already observed in previous section, that is, the model is effective in identifying bots and humans.

Scrutinising all 19 misclassified bots (***FN***) we found that ten of them were already problematic during the first step of the manual rating process, where they were rated as either "Human" or "I don't know" by one of the raters. Moreover, the final decision to classify them as bots during the discussion

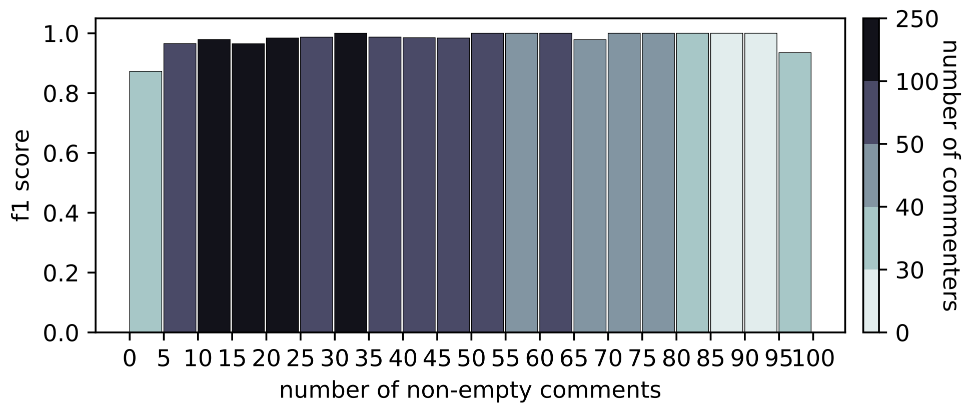


Figure 5.2: $F1$-score of the model when applied on commenters, grouped by their number of non-empty comments. The colour indicates the number of commenters in each bin.

session among raters was based on additional information that is not available in the comments themselves, explaining why the model is not able to classify them correctly.

The model also misclassified 13 humans. The fact that the model misclassified these humans as bots is not surprising given that, during the first step of the rating process, 10 out of 13 cases were manually rated as *difficult* or *very difficult*, 2 cases as "I don't know" by both raters and one case was even rated as a bot by one of the raters. Section 5.4 provides a detailed analysis of these misclassified commenters.

Since the model relies on features computed on comments to distinguish bots from humans, it is worthwhile to consider and measure the impact of the number of considered comments on the performance of the model. In particular, we aim to identify the minimal number of non-empty comments required to reliably classify bots and humans. To this end, we evaluated our model and computed the $F1$-score for commenters in the test set, grouped by their number of non-empty comments.

Fig. 5.2 shows the resulting $F1$-scores of the model grouped by bins based on the number of non-empty comments. The colour of a bin indicates how many commenters there are in that bin. The bins with 10 to 24 and 30 to 34 non-empty comments have the highest number of commenters, while bins between 85 to 94 have the lowest number of commenters. The $F1$-score increases from 0.87 (bin 0-4) and becomes stable around 0.96 to 1.00 after 10 non-empty comments are reached (from bin 10-14). This suggests that having at least 10 non-empty comments is enough to achieve good performance with the model.

5.3. The BoDeGHa bot detection tool

Since the classifier we trained to identify bots presents very good performance, we implemented it as part of a tool. The tool is called BoDeGHa (Bot Detector for GitHub activity), is developed for Python 3.7 and is easily installable through pip, the official package manager for Python.[1] BoDeGHa can be used by any researcher or practitioner to classify accounts of a given GitHub repository either as bot or as human based on their issue and PR comments.

In its simplest form, BoDeGHa accepts the name of a GitHub repository and a GitHub API key. BoDeGHa computes its output in three steps, summarized in Fig. 5.3. The first step consists of downloading all comments from the specified repository thanks to GitHub's GraphQL API. This step results in a list of commenters and their corresponding comments. The second step consists of computing the number of comments, empty comments, comment pattern and inequality between number of comments within patterns (i.e., the features of the classification model). The third step simply applies the pre-trained model on these examples, and outputs the prediction made by the model.

BoDeGHa supports several additional parameters. The minimum and maximum number of comments to download and to consider can be specified, as well as the start date from which to consider comments. It is also possible to provide a list of specific accounts for the tool to consider. To ease its reuse by other tools, it is also possible to export the results either as comma-separated values or JSON. The command-line interface of BoDeGHa is summarized in Fig. 5.4.

Fig. 5.5 presents the output of BoDeGHa for a randomly chosen GitHub repository. The output shows, for each GitHub account (first column), the number of extracted comments (second column), the number of empty comments (third column), the number of computed comment patterns (fourth column), and the inequality among them (fifth column). The last column provides the predicted class of each account. This example shows that three commenters are identified as bots, and all remaining commenters as humans.

[1]Using `pip install git+https://github.com/mehdigolzadeh/BoDeGHa`

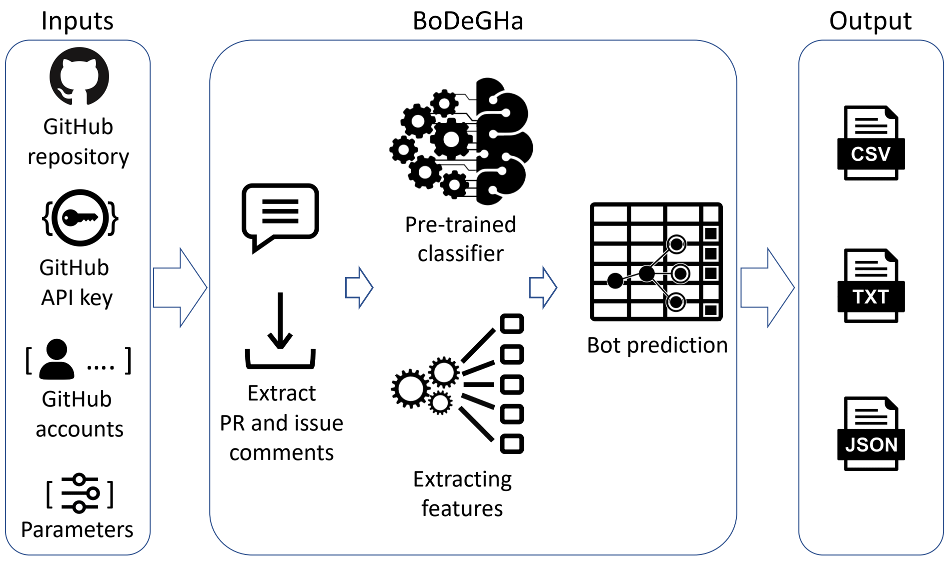
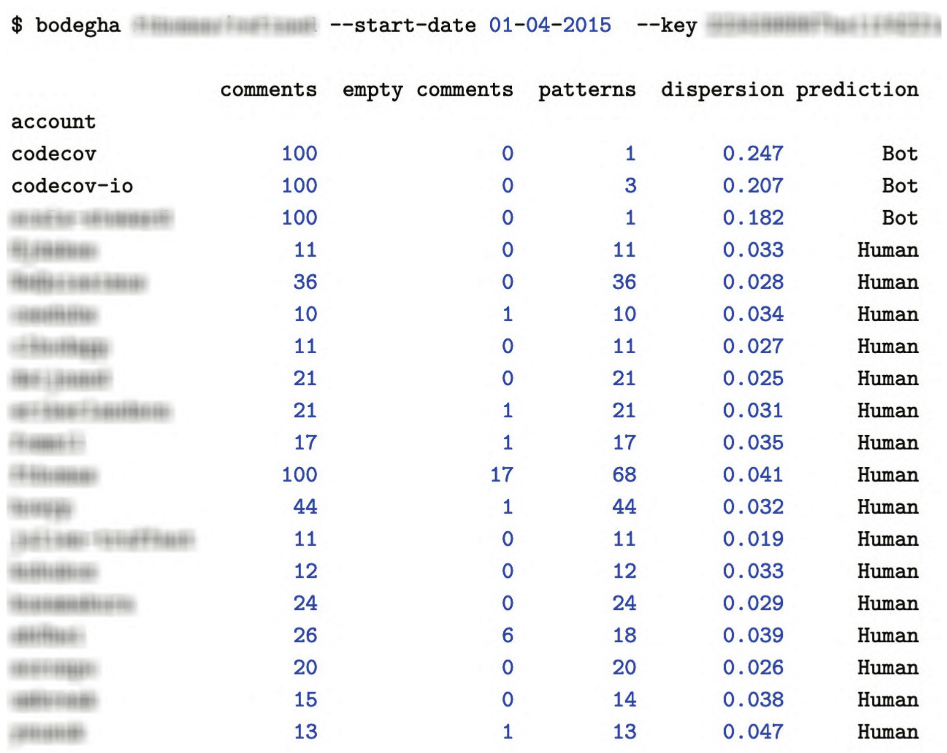


Figure 5.3: The BoDeGHa architecture.

```
$ bodegha -h
usage: bodegha [-h] repository [--accounts [ACCOUNT [ACCOUNT ...]]]
       [--exclude [ACCOUNT [ACCOUNT ...]]] [--start-date START_DATE]
       [--verbose] [--min-comments MIN_COMMENTS] [--only-predicted]
       [--max-comments MAX_COMMENTS] [--text | --csv | --json]
        --key APIKEY
```

Figure 5.4: List of command-line arguments for BoDeGHa 1.0.1.



```
$ bodegha [redacted] --start-date 01-04-2015  --key [redacted]

                 comments  empty comments  patterns  dispersion prediction
account
codecov               100               0         1       0.247        Bot
codecov-io            100               0         3       0.207        Bot
[redacted]            100               0         1       0.182        Bot
[redacted]             11               0        11       0.033      Human
[redacted]             36               0        36       0.028      Human
[redacted]             10               1        10       0.034      Human
[redacted]             11               0        11       0.027      Human
[redacted]             21               0        21       0.025      Human
[redacted]             21               1        21       0.031      Human
[redacted]             17               1        17       0.035      Human
[redacted]            100              17        68       0.041      Human
[redacted]             44               1        44       0.032      Human
[redacted]             11               0        11       0.019      Human
[redacted]             12               0        12       0.033      Human
[redacted]             24               0        24       0.029      Human
[redacted]             26               6        18       0.039      Human
[redacted]             20               0        20       0.026      Human
[redacted]             15               0        14       0.038      Human
[redacted]             13               1        13       0.047      Human
```

Figure 5.5: Example of running BoDeGHa.

5.4. Limitations of the bot detection model

The evaluation of the classifier revealed several commenters that the model was not able to properly classify. We specifically look at the commenters that have been misclassified by the model. During the evaluation of the model on the test set, we found 19 bots and 13 humans that were misclassified. In order to have a more complete categorisation of misclassified commenters, we also applied the model on the training set and obtained 4 additional bots and 12 additional humans that are misclassified.

Starting with the 24 (19+5) bots, we found that in most cases they correspond to bots that use, convert or copy text that was initially produced by humans. Even if these bots perform repetitive tasks (i.e., copy information) and even if some of these bots use templates to transfer or copy comments that are recognizable to the human eye (e.g., "*Jira issue originally created by user {username}: {content of the issue}*"), it is difficult for an automated algorithm to detect such cases.

Copy from humans (9 bots): We found some instances of bots whose comments were generated based on content made by humans (e.g., taskcat-ci, trax-robot). Since our model solely relies on features derived from comments, bot comments originating from human messages increase the likelihood of an incorrect classification. Among these cases, we found several bots that transfer data (including issues, PRs and their associated comments) to GitHub from issue trackers, code review support tools, email etc. For example, neos-bot transfers all issues from a Jira issue tracker, suchabot duplicates comments and issues from another system to GitHub, and wallabag-bot migration from email content to GitHub.

Insufficient comments (9 bots): We found 9 bots (e.g., devtools-bot and egg-bot) that were wrongly identified as humans due to the lack of a sufficient number of non-empty comments. Since our model relies on comment contents, bots with too few non-empty comments may lead to incorrect predictions even if these comments have similar comment patterns. We do not see any direct way to overcome this, since bots are expected to provide relevant information about what they are doing, and as such, one can expect their comments to be informative and non-empty.

Diverse comments (6 bots): We found 6 cases of bots that are used for the purpose of reporting, logging, or proposing code changes. The variation of comments in these bots increases the number of comment patterns, which prevents the model from identifying these bots. The source of the comment diversity comes from the reports they send for each task. For example sentry-io creates an issue each time an error occurs in the software project, along with the details of this error (e.g., stack trace). Another example is violinist-bot that submits a PR to update outdated dependencies and to report about the changes of this update. Despite these comments starting with a similar sentence (e.g., "*Sentry Issue:*" or "*If you have a high test coverage index, and your tests for this pull request are passing, it should be both safe and recommended to merge this update. Here is a list of changes between the version you use, and the version this pull request updates to:*"), they mainly consist of details related to the submitted issue or PR (i.e., stack traces for sentry-io and list of issues for violinist-bot) and are considered as different comment patterns. This prompts the classifier to consider these comments as belonging to distinct comment patterns. Misclassification of such bots could potentially be avoided by parsing the content of comments to find a template or structure.

We also looked at 17 (13+4) humans that were misclassified as bots, and created the following categories:[2]

Repetitive comments (8 humans): We found 8 instances of human commenters whose comments are mostly composed of repetitive messages, such as *thank you* or *LGTM* and that have nearly no other comments. Since repetitive messages are usually indicative of the presence of a bot, the model failed to correctly classify these commenters.

Insufficient comments (3 humans): We found 3 humans with few comments, most of them being empty. Most of these comments were created in the context of a pull request whose title was already sufficiently informative. Since these empty comments are grouped in a single comment pattern, and since they form the large majority of the comments made by these commenters, they were wrongly considered as being generated by a bot due to their repetitive nature. We also found instances where the comment content is too short or there are too few non-empty comments. This prompts our algorithm to group them into a small number of patterns and consequently provide wrong predictions.

Mostly unfilled issue templates (3 humans): It is not unusual in GitHub repositories to require commenters to follow a comment template or a check-

[2]To comply with GDPR regulations, we cannot provide the account names for these cases.

list when creating issues or pull requests.[3] We found 3 commenters whose comments were mostly composed of unfilled or barely filled templates, leading these comments to be considered as a single pattern, and leading the model to misclassify them as bots. Relying on an analysis of the content of such comments could prevent them from being misclassified, by taking into account the presence of such templates.

Others (3 humans): These cases do not fall into any of the above categories, and we have found no specific reason to explain their misclassification. Some of them have a small number of comments, while others only have a few patterns (e.g., due to the presence of similar long URLs in comments) despite the fact that they do not seem to have duplicated or similar comments.

Most commenters that were misclassified by the classification model were also hard to recognize by the raters during the process of creating the ground-truth dataset. In the test set, about 84.6% (11 out of 13) of the humans that were misclassified as bots and about 63.1% (12 out of 19) of the bots that were misclassified as humans were originally rated as "I don't know", "difficult", or "very difficult" by at least one of the raters. In contrast, among the correctly classified commenters, a much lower percentage of bots (12.5%, 24 out of 192) and humans (9.5%, 169 out of 1772) were rated as such.

Furthermore, during the creation of the ground-truth dataset, we encountered several examples of commenters whose features and comments were reminiscent of both humans and bots. Such so-called "mixed" commenters are the result of GitHub accounts belonging to humans allowing automatic tools to use their account for carrying out certain specific tasks. Hence, the comments of such commenters include both human-like and bot-like behaviour. We identified 78 such commenters out of 5,082 commenters (i.e., 1.5%) during the rating phase and we consistently excluded them from the ground-truth dataset since we could not decide whether these commenters should be classified as bots or humans.

Nevertheless, it is interesting to report how our model behaves when exposed to these specific "mixed" cases. Out of these 78 identified "mixed" commenters, 21 were classified as bots (26.9%) and 57 as humans (73.1%). The fact that the proportion of "mixed" commenters classified as bots is higher than the one in the training set (10.3%) suggests that their behaviour is perceived to be closer to that of a bot than a human by the classification model.

The presence of mixed accounts as well as the categories of bots that have

[3]See `https://docs.github.com/en/github/building-a-strong-community/about-issue-and-pull-request-templates`

been misclassified as humans suggests that it is not easy to come up with a single definition for a bot. Two persons could easily disagree on whether a given account is a bot or a human if they have a different interpretation of what it means to be bot. This calls for a more precise definition of bots. Erlenhov et al. (Erlenhov et al., 2020b) started doing so based on qualitative interviews with developers. This enabled them to identify three distinct DevBot *personas* that differ in terms of features like autonomy, chat interfaces, and smartness. This more fine-grained classification of DevBots and their characteristics paves the way for more sophisticated classification models.

The approach presented in this chapter is not the first one to have been proposed in the literature to detect bots in social coding platforms. Dey et al. (Dey et al., 2020a) proposed three different approaches for identifying bot accounts in GitHub projects, mostly based on their commit messages. One of them consists of checking for the presence of the string "bot" in the account name of the committer. We partially relied on this heuristic to add more potential bot candidates during our data collection. However, solely relying on it to identify bots is likely to lead to a large number of both false positives and false negatives. To confirm this, we applied their approach on our ground-truth dataset. We found 169 humans out of 4,473 (3.8%) containing the string "bot" in their account name, either at the end (46 cases) or in the middle (123 cases). Out of the 527 bots we have in the dataset, 394 of them (i.e., 74.7%) actually contained "bot" in their account name, usually at the end of the name (378 cases). Although this may seem high for such a simple heuristic, it still implies that more than one out of four bots is missed with this method, and about one out of 25 humans is mistakenly considered a bot. For comparison, around only one out of 25 (3.8%) bots have been misclassified as humans by our model, and around only one out of 100 humans (1.1%).

5.5. Threats to validity

Based on the structure recommended by Wohlin et al. (Wohlin et al., 2012) we discuss the threats that might call into question the validity of our findings, their potential impact and how we have tried to mitigate them.

Construct validity examines the relationship between the theory behind the experiments performed and the observations found. This threat is mainly related to correctness of the dataset used in the experiments. The results of our study are strongly dependent on the correctness of the ground-truth dataset. We are confident that the ground truth contains very few errors, since we achieved an *almost perfect* agreement ($\kappa = 0.96$) based on an iterative rating process involving all authors of the paper. One of the most likely threats is the existence of “mixed” commenters in the dataset. Such commenters are difficult to classify, even by human raters, since they combine both bot-like and human-like behaviour. Mixed commenters constitutes a very small proportion of our dataset (78 cases, corresponding to 1.5% of all considered accounts). We excluded all these cases from the dataset since we could not agree on them. However, it is possible that the dataset still contains such cases that were not identified by the raters. Given the very low ratio of such mixed accounts, it is however unlikely to affect our findings.

Internal validity concerns choices and parameters of the experimental setup that could affect the results of the observations. Given that our classification method is fully based on features computed from comments, we required each commenter included in the dataset to have contributed at least 10 (possible empty) comments. This threshold is based on the analysis in Chapter 4. As such, we cannot claim that our model applies on commenters who made fewer than 10 comments. Similarly, we considered at most 100 comments for each commenter but, as explained in Section 5.2, this upper limit on the number of comments is unlikely to have biased our results, since we already achieved high $F1$-score starting from 10 non-empty comments.

Conclusion validity concerns whether the conclusions derived from the analysis are reasonable. Our conclusions are based on the evaluation and application of the classification model on the test set. Given that we properly followed a standard grid-search cross-validation method to identify the best classifier, and that we evaluated the model on the test set (i.e., examples that have not been used to train or select the classifier), the results we obtained and conclusions we reached are unlikely to be affected.

External validity concerns the degree to which the conclusions we derived are generalisable outside the scope of this study. The main threat to external validity is related to the construction of the ground-truth dataset. To avoid any potential bias, we randomly selected a large collection of GitHub repositories related to software development and corresponding to actual packages being officially distributed, following the guidelines of Kalliamvakou et al. (Kalliamvakou et al., 2014b).

While this dataset can be regarded as representative of bots contributing to GitHub repositories through PR and issue comments, we do not make any claim about its generalisability to other activities (e.g., commit messages) or other social coding platforms (e.g., BitBucket or GitLab). Nevertheless, the underlying approach could be made applicable to such activities or platforms.

5.6. Summary and conclusions

This chapter described our novel approach to distinguish between bots and humans in collaborative software development repositories on GitHub, based on the comments they made in issues and pull requests. We developed a classification model to identify bots based on four features: the total number of comments of a commenter; its number of non-empty comments; its number of comment patterns; and the inequality between the number of comments in each pattern. The chosen features align with behavioural differences we observed between bots and humans. Indeed, we found that most human commenters tend to have diverse sets of comments with little repetition, while bots tend to frequently use a limited set of comment patterns.

The accuracy achieved by our classification model is very promising, however, there are still some limitations that must be addressed. One of the limitations is that the model only works with pull request comments and issue comments and it is not able to identify bots that are active in commit activities. Considering that the proposed model relies on message patterns, the model cannot distinguish mixed accounts, nor can it identify accounts that have had few comments in a repository. Additionally, the model and the associated tool only analyse an account in a single repository and does not consider the activities of accounts in multiple repositories. Chapter 6 will therefore propose several improvements to the classification model and its associated bot identification tool to reduce some of its shortcomings.

CHAPTER 6

Improving and extending bot detection techniques

"Knowledge is not what is memorized.
Knowledge is what benefits."

Ali ibn Abi Talibs

Chapter 5 introduced a new classification model and associated tool to identify bots in GitHub repositories. While the results of the evaluation of the classification model are promising, it does have some limitations and shortcomings. This chapter describes our efforts to address some of the limitations of our classification model.

In order to overcome the limitation of only working with pull request and issue comments, we applied the model on a dataset of commit messages and trained a new model based on the same set of features. Using the new model, we developed a tool called BoDeGiC, which we present in this chapter. Since our method is based on message similarity, it can fail in the presence of mixed accounts that combine both human and bot activities in a single account. Therefore, we propose a more fine-grained machine learning model for classifying individual issue and pull request comments using NLP techniques in order to distinguish between human and automated activities.

Building on the wisdom of the crowd principle, we propose an extension to the classification model proposed in Chapter 5 that incorporates diverging predictions from accounts active in multiple repositories, thereby leveraging data from a broader range of sources to improve the model's accuracy. Lastly, we make a comparison of different bot identification techniques that researchers have utilized in the literature and combine all of these methods to generate an ensemble classification model. We use a small set of active GitHub repositories to train and evaluate this ensemble model. The results indicate a better performance than individual methods.

6.1. A classification model based on commit messages

In Chapter 5, we developed and proposed a classification model and associated tool, called BoDeGHa to identify bots in GitHub repositories based on the comments they made in PRs and issues. To train and evaluate this model, we created a ground-truth dataset of GitHub contributors that were manually labeled as bots and humans by three raters. The evaluation of the model on the test set achieved a very high overall accuracy.

In parallel to our research, Dey et al. (Dey et al., 2020a) conducted a similar study aiming to identify bots in git repositories based on git commit data. They created a ground-truth dataset based on the World of Code (WoC) dataset (Ma et al., 2019) containing 73M Git repositories. Their ground-truth dataset is composed of 13,150 bots and 13,150 humans that were identified using BIN (for Bot Identification by Name), a technique relying on the presence of some keywords (e.g., "bot") to identify bots. This ground-truth dataset only includes true positive cases, and does not account for false negatives. Based on this ground-truth dataset, they proposed BIMAN (for Bot Identification by commit Message, commit Association, and contributor Name) to detect bots based on commit activities. BIMAN achieves an AUC-ROC of 0.90. The AUC-ROC is short for Area Under the Receiver Operating Characteristic Curve. It is a widely used performance metric in machine learning classification tasks that measures the ability of a model to distinguish between positive and negative classes. A value of 1 indicates perfect classification while a value of 0.5 indicates random classification (Fawcett, 2006).

Dey et al. used an ensemble technique composed of three methods to consider different aspects of commits made by authors. They used commit messages to identify whether a commit message generated from a template (approximately similar to our idea) called BIM. This technique achieved a precision of 0.57 and a recall of 0.67 in detecting bots. As the second method, they trained a random forest classifier to detect bots from characteristics of commits like the total number of files changed in each commit, file extensions, average file per commit, and the median number of projects the commit associated with (i.e., BICA). Applying the technique on the dataset they obtained an AUC-ROC of 0.89. And finally, they matched the author's name and email to common bot patterns (e.g., presence of the string "bot" in the author name) called BIN. This technique revealed a very high precision of 0.99 but a very low recall of 0.36.

Although both aforementioned studies pursue the same goal, they are different in essence. The most prominent difference is the type of data on the basis of which bots are being identified. PR and issue comments are messages that are being used to explain or discuss issues or PRs. Such comments are not limited in size, in contrast to commit messages that aim to provide one-liners that summarise the changes made in a commit. In Chapter 5 we identified bots based on PR and issue comments, while BIMAN relies on git commit information to distinguish bots from humans. As a consequence, the set of contributors considered in both cases is different, since contributors active in PR or issue comments are not necessarily active in code commits, and vice versa. Another difference is that we restricted our dataset to contributors active in GitHub while BIMAN works on all types of git repositories, even if they are not hosted on GitHub. Moreover, when a contributor is active in more than one repository, we considered each repository individually while BIMAN aggregates the activity from multiple repositories for each contributor.

6.1.1 Initial classification model

In Chapters 4 and 5, we proposed an approach to distinguish bots from humans based on their PR and issue commenting activities in GitHub repositories. We developed a model (and associated command-line tool) with a high precision to predict whether a contributor is a bot or human based on the comments made in issues and PRs. The underlying idea was that bots perform automated tasks, therefore, they are assumed to have more repetitive comments than humans. To capture this repetition of comments the model was trained using four features related to the comments associated to a contributor: (i) we measured the number of comment patterns on the basis of a compound comment similarity metric. We hypothesized that the less comment patterns a contributor has, the more likely a contributor is a bot; (ii) we computed the Gini coefficient to capture the inequality of the number of comments in patterns; (iii) we counted the number of comments since it allows to distinguish between contributors having a similar number of comment patterns; (iv) and we counted the number of empty comments, driven by the assumption that bots are supposed to have meaningful non-empty messages. The rationale behind these features and how we computed them are the same as what we described in Chapter 4.

To evaluate the performance of the model, we relied on a ground-truth dataset composed of 5,000 distinct GitHub contributors. To create such a ground-truth dataset, we manually labeled each contributor as bot or human with high inter-rater agreement. The final dataset contains 527 bots and 4,473 humans. We trained and compared various classification models, and achieved the highest results ($F1$-score $= 0.98$) with a random forest classifier. Not only does the model perform well in general, it also achieves high precision and recall for both classes: bots achieved a precision of 0.94, a recall of 0.94 and $F1$ score of 0.92, and humans achieved a precision of 0.99, recall of 0.99 and $F1$ score of 0.99. Only 19 out of 211 bots and 13 out of 1,789 humans were misclassified by the model.

6.1.2 Data extraction

The model we introduced in Chapter 5 performed very well to identify bots based on the repetitiveness of their comments, a text-based activity. Since git commit messages are also text-based, and since we can expect that bots active in commits exhibit a similar kind of behaviour, it seems promising to apply our model to git commit messages as well.

In this section, we will first evaluate how well our model (trained on PR and issue comments) performs when applied as-is to git commit messages. Then, we will evaluate the approach we developed in Chapter 5 applied on git commit messages, by training a new model. To do so, we need a labeled dataset of contributors and their commit messages. We rely on the dataset of git commit messages that was used by Dey et al. (2020a) and has been made publicly available.[1]

We transformed the dataset to conform to the input format required by our classification model. First of all, as explained in previous section, our model expects a set of features related to comments of a contributor in a specific repository. Therefore, if a contributor is active in more than one repository, we split its activity by repository. Our approach being based on the assumption that bots exhibit more repetitive activities, it cannot be applied for contributors that do not have enough activities. In Chapter 4, we observed that the performance of the model decreased when the number of comments was below 10. Similarly, in this classification model, we will only consider contributors that have at least 10 commit messages. We rely on 100 commit messages to compute the set of features. This upper bound significantly reduces the computational cost, and has been shown to be more than sufficient to achieve a very high precision in Chapter 5. Not imposing an upper bound would re-

[1] `https://zenodo.org/record/4042126`

Table 6.1: Summary of the dataset characteristics.

original commit dataset from (Dey et al., 2020a)	**number**
# Git repositories	6,394
# commits	311,622
# distinct contributors	6,922
↪ # bots	3,380
↪ # humans	3,542

quire to consider all commits for each contributor-repository pair, and some pairs have more than 20,000 commit messages. Even for pairs with over 1,000 commit messages the performance begins to slow down considerably.

After having performed these steps, the resulting dataset contains 311,622 commit messages from 6,922 contributors, 3,380 whom have been labeled as bots and 3,542 as humans. This accounts for around 25% of the original dataset. The dataset characteristics are summarized in Table 6.1.

6.1.3 Model and evaluation

In this section, we will first evaluate how the classification model trained on PR and issue comments performs when applied as-is to git commit messages. Then, we will evaluate how the approach developed in Chapter 5 applies to git commit messages, by training a new classification model.

We start by applying the existing model to see how it performs when applied on a new kind of data (i.e., on git commit messages). For each of the 6,922 contributors in the dataset, we computed the features required by the model, and we asked the model to predict whether the contributor is a bot or a human. We then compared the predictions with the ground-truth, enabling us to compute the precision P of the model, its recall R and its $F1$-score. The results are reported in Table 6.2.

The model achieved a precision of 0.77, with about 22.1% (749 out of 3,380) false negatives (**FN**) of bots misclassified as humans, and about 23.9% (848 out of 3,542) false positives (**FP**) of humans misclassified as bots. Most contributors are correctly classified as bot or human by the existing model even though it was not trained on git commit messages but on PR and issue comments. A possible explanation for this result is that even though the model was originally trained on issue and PR comments, it mostly captures

Table 6.2: Evaluation of the classification model of Chapter 5.

	classified as bot	classified as human	P	R	F1
Bot	2,631 (TP)	749 (FN)	0.76	0.78	0.77
Human	848 (FP)	2,694 (TN)	0.78	0.76	0.77
weighted avg			**0.77**	0.77	0.77

the repetitive nature of tasks. Therefore, it shouldn't be that surprising that it also works on commit messages, where we can also expect bots to have repetitive behaviour.

To see how the approach developed in Chapter 5 behaves on git commit messages, we trained a new model on git commit messages as opposed to the previous model that was trained on PR and issue comments. To do so, we divided the ground-truth dataset into two disjoint subsets. 60% of the data are used to perform grid-search cross-validation to select the best classifier and its parameters. A test set composed of the remaining 40% is used to evaluate the selected classifier on unseen data. At the end of the cross-validation set, we obtained 91 different classifiers. The performance of these classifiers was measured using traditional performance metrics of precision P, recall R, and $F1$-score for the population of each class (i.e., for bots B and human H). We report the highest $F1$-score for each classifier in Table 6.3, in descending order. We retained the random forest (RF) classifier, as it slightly outperforms the other classifiers. Its score was obtained with the *entropy* split criterion, 20 estimators (i.e., trees) and a tree depth of 8.

Table 6.3: Precision, recall and $F1$ of the best-performing classifiers per classifier family (in descending order of $F1$).

	bots		**humans**		**overall** ($B \cup H$)		
classifier family	$P(B)$	$R(B)$	$P(H)$	$R(H)$	P	R	$F1$
random forest (**RF**)	0.817	0.748	0.775	0.837	0.817	0.748	0.793
decision trees	0.845	0.698	0.750	0.876	0.845	0.698	0.787
SVM	0.798	0.735	0.762	0.819	0.798	0.735	0.777
logistic regression	0.807	0.720	0.755	0.832	0.807	0.720	0.776
k-nearest neighbours	0.831	0.653	0.722	0.872	0.831	0.653	0.761

We evaluated the selected classifier on the test set containing the remaining 40% data. This test set includes 2,769 identities, of which 1,417 correspond to humans and 1,352 correspond to bots. The evaluation results are reported in Table 6.4. With this retrained model about 24.6% of bots (333 out of 1,352) are misclassified as humans (**FN**), and about 15.9% of humans (226 out of 1,417) are misclassified as bots (**FP**). Compared to the previous model, the model trained on commit messages detects humans more accurately, while the converse can be observed for bots. With a value of 0.80, the precision of the retrained model is slightly higher than the previous one.

Table 6.4: Evaluation of the retrained classification model.

	classified as bot	classified as human	P	R	F1
Bot	1,019 (TP)	226 (FP)	0.82	0.75	0.78
Human	333 (FN)	1,191 (TN)	0.78	0.84	0.81
weighted avg			**0.80**	0.80	0.80

6.1.4 The BoDeGiC bot detection tool

In order to enable practitioners to use the classification model, we implemented it through BoDeGiC (Bot Detector for Git Commits),[2] a command-line tool to detect bots in given git repositories. The tool analyses the commit messages of each contributor in the specified git repositories and predicts whether the contributor is a bot or a human. BoDeGiC is implemented in Python 3.7 and is easily installable through pip, the official package manager for Python.

BoDeGiC works in three steps. The first step consists of extracting all commit information from the specified Git repository using `git log`. This step results in a list of commits, authors and their corresponding commit messages. The second step consists of computing the features to feed the classification model. Features consists of the total number of messages, the number of empty messages, the number of message patterns and the inequality between the number of messages within patterns. In the third step, we apply the classifier that was trained on commit messages to the extracted data. The tool outputs the prediction made by the model for each contributor.

[2]`https://github.com/mehdigolzadeh/BoDeGiC`

Listing 6.1: List of command-line arguments for BoDeGiC 0.2.0.

```
$ bodegic -h
usage: bodegic [-h] [--include [NAME [NAME ...]]]
[--start-date START_DATE][--mapping [MAPPING]][--verbose]
[--min-commits MIN_COMMITS][--committer][--max-commits MAX_COMMITS][--text |
    --csv | --json][REPOSITORY [REPOSITORY ...]][--only-predicted]
```

The command-line interface of BoDeGiC is summarized in Listing 6.1. The output and the behaviour of BoDeGiC can be adapted by many optional command-line arguments in several different ways. By default, BoDeGiC relies on the author names in git commits, but the committer names can be used instead by specifying `-committer`. The list of names to consider can be explicitly specified with `-include`. BoDeGiC also supports identity merging (i.e., when a contributor uses multiple names) through the `-mapping` parameter. This parameter expects a path to a CSV file specifying how to map names to identities. This file can also be used to ignore specific names, by mapping them to the special "IGNORE" identity. Since the model (and by extension, the tool) requires at least 10 commits for a contributor to generate a prediction, contributors that have commits less than this number are predicted as "Unknown". These cases can be excluded from the output by means of the `-only-predicted` parameter. Additionally, The minimum and maximum number of commits to consider can be changed respectively with `-min-commits` and `-max-commits`. By default, BoDeGiC outputs one line per author with the predicted class. The set of computed features can be included in this output by adding the `-verbose` parameter. Finally, the output of BoDeGiC can be exported in text (by default) or in JSON or as a CSV.

Fig. 6.1 presents the output of BoDeGiC on a randomly chosen Git repository that was analysed on 2020-10-14. The first column shows the contributor name, the second column the number of extracted commit messages, the third column the number of computed message patterns, and the fourth column the statistical dispersion of the number of comments per pattern as computed by the Gini inequality index. The last column provides the predicted class of each contributor.

6.1.5 Discussion

The main threats to validity and mitigation strategies mentioned in Section 5.5 also apply to the current study.

A distinct threat to construct validity stems from the ground-truth dataset that has been used to train and evaluate the models. The dataset was created by other researchers and we have assumed it was correctly built and validated. Any presence of mislabeled items in that dataset could negatively affect the results of applying our original classification model (i.e., that was trained on PR and issue comments) as well as the retrained model that was directly trained on this dataset.

In order to assess to what extent this threat holds, we selected a subset of contributors from the dataset and manually verified whether they are actually humans or bots. We randomly selected 25 contributors from each category of correctly and incorrectly classified contributors (i.e., from TP, TN, FP and FN). Each of these cases was independently evaluated and labeled by the three authors of the paper. At the end of this process, we compared the labelings and observed agreement on all 100 cases. We then compared our own labelings with the actual labels found in the dataset, and observed a disagreement for 19 out of 100 cases. Among these cases, 13 corresponded to bots and 6 to human contributors. The prediction made by our classification model for these 19 cases matched our own labeling, i.e., the model was able to correctly predict them.

While manually looking at some other contributors in the dataset, we encountered a few cases we could not agree on because they combine both bot-like and human-like behaviours. We already encountered such "mixed" contributors in Chapter 5 where we found that some contributors were occasionally relying on tools or bots to automate part of their activities. The presence of such mixed cases in git commit messages reinforces our belief that a better definition of "what a bot is" is required, with a clearer boundary between humans and bots. We also believe, in view of these mixed cases, that it might be interesting to identify bots not at the level of a contributor but at the level of its activities. In other words, the question "Is this contributor a bot?" would become "Is this contributor activity produced by a bot?".

```
$ bodegic [redacted] --verbose --committer

                  messages patterns  dispersion prediction
committer
Travis CI[bot]          20        1       0.026        Bot
greenkeeper[bot]        10        3       0.141        Bot
[redacted]              69       58       0.040      Human
snyk-bot                 5      NaN         NaN    Unknown
```

Figure 6.1: Example of running BoDeGiC (version 0.2.0).

6.2. A classification model to identify bot activities

Although the previously mentioned approaches are useful to identify bots at the account level, an account-level classification does not always suffice. Such a classification can fail in the presence of mixed accounts. While creating the ground-truth that we introduced in the previous chapter, we identified 78 accounts out of 5,082 GitHub accounts combining both activities. This happens when users grant bots access to post comments on their behalf (e.g., semantic-release bot). The granularity of account-level classifications is insufficient to differentiate between human and bot activity at the level of individual comments.

Even in cases where an account is predominantly producing bot (or human) comments, mixed activity may still be observed. For example, human developers may manually produce comments on behalf of a bot when testing this bot in its early stages of adoption. The above observations call for the need for a more fine-grained classification that is able to identify bot or human activity at the level of individual comments as opposed to the account level.

In this section we propose a classification model to identify bot activity at the level of individual comments. To achieve this, we first transform raw text into machine-understandable features using natural language preprocessing and encoding. Next, we select among a list of machine learning binary classifiers the best performing one in order to classify each comment as bot or human. The main value of this approach is the ability to classify comment as originating from a bot or a human without requiring to analyse the entire account's commenting activity. This means that comments can be classified fast, and large datasets can be analysed efficiently.

6.2.1 Data extraction

In order to train and evaluate a model aiming at distinguishing GitHub comments created by bots from comments created by humans, we need a large dataset of such pre-labelled comments. In Chapter 4, we created a ground-truth dataset of 5,000 accounts that were manually identified by at least two raters as bot or human based on their PR and issue comments. This dataset[3] contains 28,287 comments made by 527 bots and 268,504 comments made by 4,473 humans, from which mixed accounts were excluded.

For this study, we extracted from this dataset a random, balanced subset of 19,282 comments, composed of 9,641 comments created by 519 bots, and

[3]The dataset is available on `http://doi.org/10.5281/zenodo.4000388`.

9,641 comments created by 4,090 humans.

6.2.2 Model construction

In this section, we propose a machine learning model for classifying GitHub PR and issue comments. The preprocessing part of the model combines two widely used techniques in natural language processing for encoding the comment text into machine-understandable features. The classification part of the model takes as input the preprocessed text to perform the classification task. In the following paragraphs, we will explain how we constructed the model, which classifier we selected, and how we tuned the parameters to get the best performance and accuracy.

Since human-written texts have no direct meaning for machine learning algorithms, natural language processing (NLP) is needed to convert such texts into a numerical representation that can be analysed by machines. A range of different methods can be used to convert raw texts into numerical vectors, such as bag-of-words (Manning et al., 2008), Term Frequency - Inverse Document Frequency (TF-IDF) (Sparck Jones, 1972), Word2Vec (Mikolov et al., 2013), and Bert (Devlin et al., 2019).

We tested all these preprocessing techniques and their variants and we achieved the highest accuracy with bag-of-words and TF-IDF. The bag of words technique creates a vector that has as many dimensions as the text corpus has unique words. If a text contains a specific word from the corpus, it will be marked as 1 in the corresponding position of the vector, and 0 otherwise. TF-IDF is similar except that it assigns a higher weight to both high and low-frequency terms in the document, and the frequency of each term is considered as the indicator of its importance. Given a comment, we pull out only the unigram words to create an unordered list of words using the bag-of-words method. Then, TF-IDF is used to form a feature vector, where each feature is a term (i.e., word) and the value of the feature is the weight of the term.

For the classification part of our model, we restrict ourselves to binary classifiers since the goal is to classify comments as being produced either by a bot or a human. We evaluated a range of binary classifiers: the ZeroR baseline classifier, Support Vector Machines (SVM) (Gunn, 1998), multinomial Naive Bayes (NB) (Lindley, 1990), Random Forest (RF) (Breiman, 2001) and k-Nearest Neighbours (kNN) (Aha et al., 1991). Since the effectiveness depends on specific input parameters, we followed a standard hyper-parameter tuning process using grid-search cross-validation (Witten et al., 2011).

We split the comment dataset of Section 6.2.1 into a training and a test set (see Table 6.5). The training set will be used as a validation set in a grid-search cross-validation process to determine the best input parameters, the best classifier and to train the selected model. The test set will be used to evaluate the performance of the selected model on new data. We created both sets so that approximately half of all bot comments (respectively human comments) belong to the training set and the other half belong to the test set.

Table 6.5: Number of bot comments and human comments in the training and test set.

	# human comments	**# bot comments**	**total**
training set	4,789	4,791	9,580
test set	4,852	4,850	9,702
total	9,641	9,641	19,282

While creating both sets, we ensured that comments belonging to the same account were not spread in both sets. The rationale is that, since comments produced by the same account are more likely to be similar or related, distributing such comments over both sets might lead to unrealistic evaluation results. More precisely, it could lead the model to be trained on specific words or combinations of words used by a commenter, hence artificially improving the evaluation results if these combinations are also present in the test set.

The performance of the resulting models is measured using the precision P, recall R and F_1 score for the population of each class (i.e., for bot comments B and human comments H). We aim to achieve an as high overall F_1 score as possible.

We rely on the default parameters for the preprocessing and encoding steps. We use grid-search cross-validation to find the best classifier and its parameters. We follow a stratified group k-fold cross-validation process to ensure that each fold preserves the proportion of bot and human comments, and that comments by the same account are not spread across folds.

Table 6.6 reports on the performance for each of the classifiers, in descending order of F_1. Only the classifier instances whose parameters resulted in the highest F_1 score within each family of classifiers are shown in the table.

We observe that all classifiers substantially outperform the ZeroR baseline classifier; this is a sanity check as this classifier is merely used as a benchmark for other classification methods. In terms of overall performance, NB and SVM appear to be the most promising classifiers. The SVM, RF, and KNN

Table 6.6: Precision, recall and F_1 score of the best classifiers per family of classifiers (in descending order of F_1).

	bot comments		**human comments**		**overall** $(B \cup H)$		
classifier	$P(B)$	$R(B)$	$P(H)$	$R(H)$	P	R	F_1
NB	0.864	0.883	0.881	0.861	0.873	0.872	0.872
SVM	0.971	0.718	0.777	0.979	0.874	0.848	0.845
RF	0.898	0.542	0.672	0.935	0.785	0.739	0.727
kNN	0.993	0.369	0.613	0.997	0.803	0.683	0.648
ZeroR	0.200	0.400	0.299	0.600	0.249	0.499	0.332

classifier have high recall $R(H)$ for human comments (0.979, 0.935 and 0.997, respectively) but a rather low recall $R(B)$ for bot comments (0.718, 0.542 and 0.369, respectively). The NB classifier, which was obtained using $\alpha = 1.5$ and uniform class prior probabilities, has the highest recall for bot comments $R(B) = 0.883$ and its overall precision, recall and F_1 score is the highest of all considered classifiers.

6.2.3 Model evaluation

We selected the best classifier NB with the parameters explained in the previous section, and trained it on the 9,580 comments of the training set. We evaluated this classification model on the 9,702 new and unseen comments of the test set, of which 4,852 are human comments and 4,850 are bot comments (cf. Table 6.5). Table 6.7 reports the evaluation results.

Table 6.7: Evaluation of the Naive Bayes classification model on the test set.

	comments classified as				
	bot	**human**	P	R	F_1
bot	TP: 4,382	FN: 468	0.866	0.904	0.884
human	FP: 680	TN: 4,172	0.900	0.860	0.880
average			0.882	0.882	0.882

The results show that around 90% of bot comments (4,382 out of 4,802) and 86% of human comments (4,172 out of 4,713) are classified correctly. The overall F_1 score is notably good (0.882), indicating that the model generalizes well on unseen data. The higher recall of bot comments (0.904) and higher precision of human comments (0.900) indicate that the model performs better in detecting bot comments. Nevertheless, we observe a decent F_1 score for both classes ($F_1(B) = 0.884$ and $F_1(H) = 0.880$), indicating the overall good performance of the model.

Manual inspection of a sample of misclassified comments revealed that these comments are difficult to classify, even for a human evaluator. For example, the following human comment was misclassified as a bot comment: "*Closing this as resolved by #72*". Conversely, the following bot comments were misclassified as human comments: "*Here are some suggestions: At index: 33, offset: 6, reason: "it was" is wordy or unneeded*" and *"If the machine has enough cores, then the work done by the babel loaders can be parallelized to run much faster."*

The selected multinomial Naive Bayes classifier has proven to show a good performance in text classification problems (Mccallum & Nigam, 2001). The decision function in this classifier predicts based on the probability computed for each case (i.e., each considered comment). If the probability value is above 0.5, then the corresponding case belongs to the target class (in our study, a bot comment) otherwise, it belongs to the complement class (in our study, a human comment). We gain deeper insight into the predictions made by the model by looking at the associated probabilities. To do so, we extracted the associated probability for test cases. To some degree, these probabilities correspond to a confidence score: a probability close to 1 indicates that there is a high level of confidence in classifying the comment as bot comment. Conversely, a probability close to 0 indicates high confidence in classifying the comment as a human comment. A probability close to 0.5 indicates low confidence in the prediction.

Fig. 6.2 shows the probability of each prediction, distinguishing between bot and human comments, by means of boxen plots (Hofmann et al., 2011). The misclassified cases correspond to those bot comments for which the probability is below 0.5 (upper left of the figure) and those human comments for which the probability is above 0.5 (lower right of the figure).

We observe that most comments have an associated probability close to 0 for human comments and close to 1 for bot comments, indicating high confidence in the prediction. Distinguishing between correctly and incorrectly classified cases, we found that bot comments that were correctly classified as such exhibit a median probability of 0.97 (i.e., closer to 1 than 0.5, high confidence), while bot comments that were misclassified as human comments have a median probability of 0.37 (i.e., closer to 0.5 than 0, low confidence). A similar observation can be made for human comments: correctly classified human

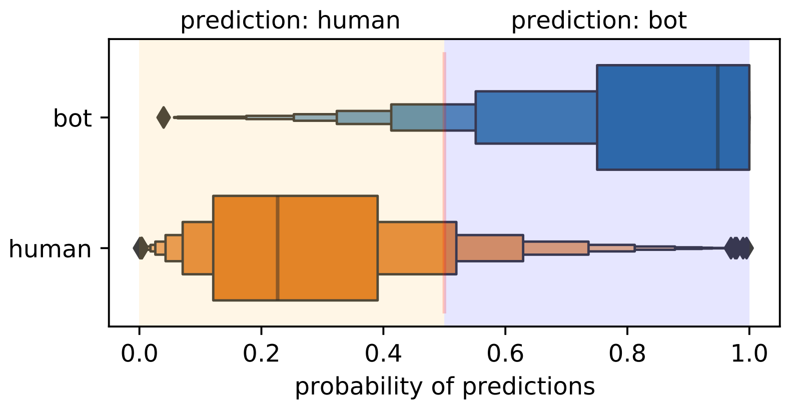


Figure 6.2: Distribution of the prediction probability for bot comments and human comments

comments have a median probability of 0.20 (i.e., closer to 0 than 0.5, high confidence), while the misclassified ones have a median probability of 0.61 (i.e., closer to 0.5 than 1, low confidence). This indicates a higher confidence for the correctly classified comments than for the misclassified ones, suggesting that a probabilistic model is more informative than a simple binary classification model to decide whether a specific comment is produced by a bot or a human.

6.2.4 Discussion

One of the main motivations of the classification model we introduced in Section 6.2 is to identify comments as having been produced by bots or human. For accounts with mixed activity, such a classification model will be able to split the bot activity from the human activity of the account. To see whether our model is able to split the bot activity from the human activity of mixed accounts, we applied it on a small dataset provided by a research group from the Eindhoven University of Technology who were investigating mixed accounts (Cassee et al., 2021). Each mixed account comment was labelled as human or bot by at least two researchers of the group, and the dataset consists of 177 bot comments and 203 human comments with full agreement. Our model was able to correctly classify about 80% of the cases. Only 3 out of 177 bot comments were misclassified as human comments and 76 out of 203 human comments were misclassified as bot comments. The model achieved an overall $F_1 = 0.78$ with an overall recall $R = 0.79$.

The performance of the model on mixed account comments is promising for future work, but the identification of human comments in mixed accounts needs to be improved. One possibility would be to train the model on a dataset that includes mixed accounts as well. Indeed, the activity of mixed accounts can be different from the one in human-only and bot-only accounts. However, this would require a substantially larger dataset than the one we have.

In Chapter 5 we also encountered bots whose comments were partly copied from humans and vice versa. For example, a translator bot followed a pattern like *"Translation from: <translation of some text>"* in their comments. We refer to these comments as mixed comments since they are composed of both human text and bot text. As a follow-up study, it would be interesting to explore how we can use or adapt the classification model to detect these mixed comments and to extract the human and bot parts of these comments.

The presence of mixed accounts and mixed comments indicates that it is not easy to characterise exactly what a bot comment is. Two different individuals could disagree on the interpretation of comments as being produced by bots. As a consequence, a more fine-grained classification of (types of) bots would be needed, building further on the work by Erlenhov et al. (Erlenhov et al., 2020a) who identified three types of bot personas based on their autonomy, chat interface, and smartness. It would be definitely interesting to explore how the binary classifier we propose can be generalised to detect these different types of bot personas.

6.3. A classification model based on multiple repositories

Identifying the presence of bots is not only useful for researchers conducting socio-technical studies but also for practitioners and funding organizations to identify contributors and to accredit them. The research literature already lists a few approaches to identify bots in software repositories, such as BIMAN (Dey et al., 2020a), BoDeGHa (Section 5.3) or BoDeGiC (Section 6.1.4). BIMAN (Dey et al., 2020a) combines three different approaches to recognize bots in commits: (i) the presence of the string "bot" at the end of the author name, (ii) repetitive commit messages, and (iii) features related to files changed in commits. BoDeGHa analyses comments posted in issues and pull requests to detect bots, based on the assumption that bots tend to frequently use a limited set of comment patterns. BoDeGic transposes this approach to commit messages, assuming that bots tend to have a limited set of commit message patterns. In Section 6.2 we proposed a probabilistic model based on

NLP techniques to detect bot activity at the level of individual comment in issues and pull requests.

In this section, our goal is to enhance the detection of bots active in issue and pull request comments, building on BoDeGHa that was introduced in Chapter 5. BoDeGHa predicts for each contributor with enough activity in the repository whether this contributor corresponds to a *bot* or a *human* contributor. If a contributor has not made enough comments, BoDeGHa classifies it as *unknown*.

Although BoDeGHa has been shown to perform well in detecting bots, it may still wrongly classify some contributors. Because BoDeGHa works at the repository level, this means that a same contributor active in multiple repositories may lead to diverging predictions, this is, it may be classified as *bot* in some repositories and as *human* in some other ones. For example, while BoDeGHa identifies the well-known *dependabot* bot correctly in many different repositories, it identifies it as a *human* contributor in *artichoke/rand_mt* because the 24 comments made by *dependabot* in this repository exhibit 10 different comment patterns, corresponding to the behaviour usually observed for human contributors. At the same time, BoDeGHa classifies the same bot as *unknown* in *cossacklabs/themis* because it only has 9 comments in this repository. Similarly, a human contributor can be sometimes classified as a bot. For example, in the GitHub repository *rust-lang/libc* we found a human contributor[4] that is detected as *bot* because most of his/her comments follow a single comment pattern of the form "*bors r+*". On the other hand, this contributor is correctly classified as *human* in *crossbeam-rs/crossbeam* and *rust-lang/rust* for example.

In this section, we investigate how frequently such situations occur in GitHub repositories. We quantify how frequently do contributors have diverging predictions (that is, predicted as *bot* and *human* by BoDeGHa), and how frequently they have incomplete predictions (that is, predicted as *unknown* by BoDeGHa). We provide preliminary insights on a novel approach to improve the accuracy of BoDeGHa by leveraging predictions from multiple repositories. We evaluate to which extent diverging and incomplete predictions can be fixed based on the wisdom of the crowd principle. More specifically, we address the following research questions:

RQ0: How frequently are contributors active in multiple repositories? We observe that one third of the contributors are active in multiple repositories.

RQ1: How frequently do contributors have diverging or incomplete predictions? More than half of the contributors identified at least once as bots have diverging

[4]Name is hidden to comply with GDPR regulations.

or incomplete predictions.
RQ2: To which extent can we fix diverging predictions? We show that an approach based on the wisdom of the crowd principle is effective at fixing diverging predictions.
RQ3: To which extent can we complete predictions? We show that the same approach is promising to address incomplete predictions.

6.3.1 Data extraction

The BoDeGHa bot identification tool takes as input a GitHub repository and outputs whether the contributors in this repository correspond to bot or human contributors. Since our goal is to improve the performance of BoDeGHa by leveraging predictions from multiple repositories, we need a large collection of GitHub repositories having their contributors active in multiple repositories. Following the advice of Kalliamvakou et al. (Kalliamvakou et al., 2014b) we need to avoid repositories that have been created merely for experimental or personal reasons, or that only show sporadic traces of activity. Good candidate datasets are collections of repositories associated to the collaborative development of open-source software packages for specific programming languages.

We collected the GitHub repositories associated with the software packages that are distributed through the Cargo package manager, for the Rust programming language. In October 2021, 68,621 Rust packages were available on Cargo and 38,886 of them (i.e., 56.7%) have an associated repository on GitHub. Since we need bots to be active in the repositories to conduct our empirical study, and since bots are more likely to be present in larger and more mature projects, we excluded packages that do not even refer to their homepage or to their documentation. This left us with 22,156 packages. Given that BoDeGHa relies on the comments made in issues and pull requests to identify bot contributors, we excluded repositories having less than 100 issues or pull requests. At the end of the data extraction process, the dataset contains 1,039 GitHub repositories accounting for 147,426 pairs of contributor/repository.

6.3.2 How frequently are contributors active in multiple repositories?

Since we aim to improve bot detection by leveraging predictions made on multiple repositories, we need contributors to be active in more than a single repository. This question aims to quantify how frequently contributors are active in multiple repositories. The 147,426 pairs of contributor/repository in

Table 6.8: Number and proportion of contributors in function of the number of repositories they are active in

# repositories →	1	2	3	4 or 5	6 - 9	10+
# contributors	5,671	1,530	496	385	239	211
% contributors	66.5%	17.9%	5.8%	4.5%	2.8%	2.5%

our dataset correspond to 57,757 distinct GitHub accounts, already indicating that some contributors are active in more than one repository. Only 8,532 contributors out of these 57K (14.8%) have enough commenting activity in at least one repository for BoDeGHa to be applied. For each of these 8,532 contributors (i.e., each distinct GitHub account), we counted the number of repositories that each contributor was active in. Table 6.8 reports on the number and proportion of contributors in function of the number of repositories they are active in.

We observe that while most contributors (5,671 out of 8,532, 66.5%) are active in a single repository only, around one third of the contributors (2,861, i.e., 33.5%) are active in multiple repositories. We will focus on those 2,861 contributors since they correspond to those for which BoDeGHa will produce several, potentially diverging (i.e., *bot* and *human*) or incomplete (i.e., *unknown*) predictions. These 2,861 contributors are active in a total of 1,010 distinct repositories.

6.3.3 How frequently do contributors have diverging or incomplete predictions?

We applied BoDeGHa on each of the 1,010 repositories identified in *RQ*0 in order to get the predictions for each of the 2,861 contributors active in two or more repositories. Under the hood, BoDeGHa downloads up to 100 pull request or issue comments for each contributor active in the repository. Only the comments made during the last five years (i.e., after December 2016) are considered. BoDeGHa then analyses these comments and predicts whether the contributor corresponds to a *bot* or a *human* contributor based on several features including the repetitiveness of comments and the number of comment patterns. If a contributor has less than 10 comments, BoDeGHa classifies it as *unknown*. At the end of this process, we have a total of 41,542 predictions of which 1,146 correspond to *bot*, 10,227 to *human* and 30,169 to *unknown*. The high proportion of *unknown* predictions (73%) indicates that most contributors have less than 10 comments in the considered repositories.

Since our focus is on improving bot detection, we select contributors that were classified *bot* at least once. Out of the initial 2,861 distinct contributors active in at least two repositories, 229 (8%) were classified *bot* at least once. Among them, 106 (46%) were consistently classified *bot* in all the repositories they were active in. Out of the 123 remaining contributors having been predicted as *bot* at least once, 60 have diverging predictions (i.e., they were also classified as *human*) and 63 have consistent but incomplete predictions (i.e., they were also classified as *unknown*).

To assess to which extent bot detection can be improved by leveraging predictions from multiple repositories, we need to determine the correct type (i.e., bot or human) of each account. Two co-authors of the paper (Chidambaram et al., 2022) manually and independently checked the 3,086 predictions for the 229 contributors that were at least once predicted as bot to determine their actual type, following an inter-rater agreement process. The first step of this process ended up with an agreement on 95% of the cases. The remaining ones were discussed together, ending up with an agreement on all of them. With this process, we found that BoDeGHa incorrectly predicted *bot* in 110 cases and incorrectly predicted *human* in 31 cases. Table 6.9 summarizes the number of actual bot and human contributors we found, as well as the number of *bot*, *human* and *unknown* predictions obtained for them.

Table 6.9: Number of actual bot and human contributors, and their number of *bot*, *human* and *unknown* predictions

		predictions		
	contributors	# *bot*	# *human*	# *unknown*
actual bot	142	1,110	31	413
actual human	87	110	288	1,134
total	229	1,220	319	1,547

6.3.4 To which extent can we fix diverging predictions?

Our result in Sections 6.3.2 and 6.3.3 revealed that many contributors have different predictions depending on the repository BoDeGHa is applied on. In this section, we propose an approach based on the wisdom of the crowd principle to fix these diverging predictions. More specifically, if one assumes that BoDeGHa is more often correct than wrong in predictions, then, given a contributor having multiple predictions, we can assume that the most frequent prediction (either *bot* or *human*) is correct, while the less frequent one is not. Let WoC-P be such bot detection model. WoC-P stands for *Wisdom of the Crowd principle for Predictions* and works on top of BoDeGHa by automatically replacing the less frequent predictions of a contributor with the most frequent ones. Ties are arbitrarily resolved as *human*.

We applied both BoDeGHa and WoC-P on the 84 contributors that have at least two predictions of which one is *bot*. Figure 6.3 shows, for each contributor, the number of *human* predictions, the number of *bot* predictions, and whether it is an actual bot or human. To permit distinguish overlapping points, we added a jitter of 0.25 on both axes. The diagonal line illustrates the WoC-P model: any contributor above the line will be consistently predicted as a bot (i.e., the *human* predictions are replaced by *bot* predictions), while any contributor below will be consistently predicted as a human (i.e., the *bot* predictions are replaced by *human* predictions).

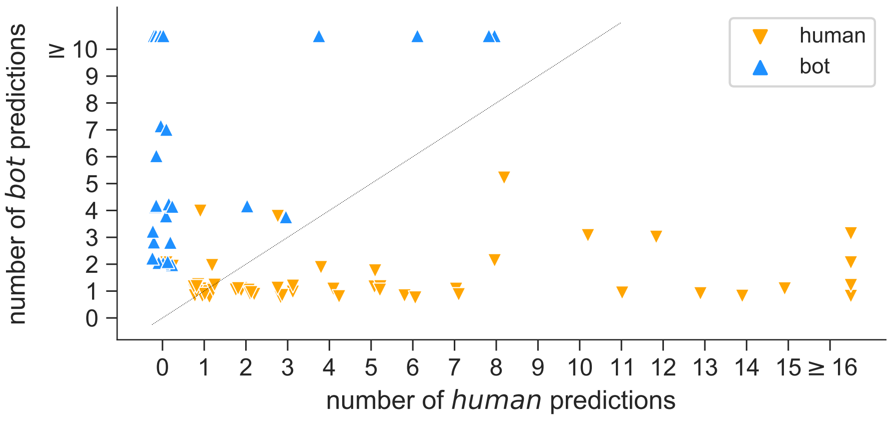


Figure 6.3: Number of *bot* and *human* predictions, each point is a contributor

As can be observed from the figure, the approach proposed by WoC-P seems promising, most of the contributors having mostly predictions corresponding to their actual type. Only five human contributors have a higher number of *bot* predictions than *human* predictions. These contributors will be consistently but wrongly predicted as *bot* by WoC-P.

To assess to which extent BoDeGHa can be improved by WoC-P, we evaluated both models on the 84 contributors. Table 6.10 reports on the resulting number of true positives (TP), true negatives (TN), false positives (FP), false negatives (FN) as well as on the accuracy (Acc), precision (Prec), recall and F1 scores of the two models.

Table 6.10: Score comparison between BoDeGHa and WoC-P

	TP	TN	FP	FN	Acc	Prec	Recall	F1
BoDeGHa	928	288	79	31	91.7	92.2	96.8	94.4
WoC-P	959	348	19	0	98.6	98.1	100.0	99.0

We observe that WoC-P actually improves the predictions made by BoDeGHa. WoC-P replaced a total of 101 predictions out of 1,326 (i.e., 7.6%): 65 *bot* predictions were correctly converted to *human* predictions, while 36 *human* predictions were converted to *bot* predictions, among which 31 correspond to actual bots. This leads the number of false negatives to drop from 31 to 0, and the number of false positives to decrease from 79 to 19. These 19 incorrect predictions correspond to the five human contributors above the diagonal line in Figure 6.3. As a consequence, WoC-P has higher accuracy, precision, recall and F1 scores compared to BoDeGHa.

6.3.5 To which extent can we complete predictions?

So far, we relied on the wisdom of the crowd principle, using the most frequent prediction to fix the less frequent predictions. This section aims to determine whether a similar approach can be followed to fix *unknown* predictions as well.

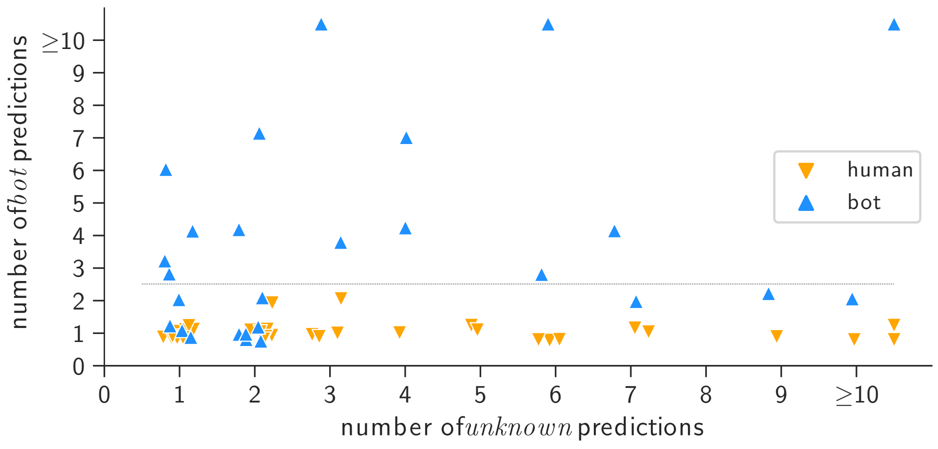


Figure 6.4: Accounts predicted as bot/unknown vs ground truth

Figure 6.4 shows the number of *unknown* and *bot* predictions for the 63 contributors that were either predicted *bot* or *unknown* (i.e., that have no *human* predictions). We observe that the situation is more delicate than for $RQ2$. Indeed, many actual human contributors are among the contributors having only *bot* and *unknown* predictions. Converting the *unknown* predictions to *bot* predictions for these 33 human contributors would only increase the number of incorrect predictions for them. For instance, while converting the 184 *unknown* predictions of the 30 bots increases the number of correct predictions from 336 to 520, doing the same for the 158 *unknown* predictions of the 33 human contributors increases the number of incorrect predictions from 35 to 193.

Nevertheless, we observe that most of these human contributors have a low number of *bot* predictions compared to the actual bot contributors. For instance, there are 17 bots and no human having three or more *bot* predictions. On the other hand, all human contributors and "only" 13 bots have one or two *bot* predictions. Converting only the *unknown* predictions of contributors having three or more *bot* predictions would increase the number of correct predictions from 318 to 460, without increasing the number of incorrect predictions. However, since this threshold of "3+ *bot* predictions" is obtained by observation, it cannot be integrated into the WoC-P model without prior validation on another dataset.

6.4. Accuracy of bot detection techniques

The increasing presence and activity of bots in software repositories makes it challenging for software engineering researchers to study socio-technical aspects of software development since their findings may be biased by not explicitly considering the presence of bots among the contributors. Similarly, it may be important for contributors that their contributions are properly recognized and rewarded since collaborative software development activities are often considered as a criterion for employers when hiring developers (Hauff & Gousios, 2015). This is especially important when funding or donations are awarded to contributors based on their contributions. While there are tools such as *SourceCred*[5] to support communities in automatically measuring and rewarding value creation, they do not automatically identify bots and their activities so far.

This is where bot identification tools come to the rescue. Such tools aim

[5]`https://sourcecred.io`

to distinguish bots from humans in GitHub accounts on the basis of their behaviour. Dey et al. (Dey et al., 2020b) proposed an automatic method to identify bot accounts in git projects based on (i) the presence of the string "bot" at the end of the author name, (ii) commit messages, and (iii) features related to files changed in commits and projects the commits are associated with. In Chapter 5 we proposed an approach and tool to detect bots in GitHub repositories based on the repetitiveness of their comments in issues and pull requests. The approach was further extended to git commit messages in Section 6.1.

This section presents an exploratory study on the accuracy of 5 bot detection techniques on a set of 540 accounts from 27 GitHub projects. We show how prevalent bots and their activities are, and that none of the bot detection techniques are accurate enough to detect bots even among the most active contributors. This highlights the importance of considering them when conducting socio-technical studies or when attributing contributions, and underlines the need for improved bot detection techniques.

In addition, we propose an ensemble model that combines these techniques to enhance the accuracy and recall of bot detection. This new model, which we refer to as EnsBoD, utilizes a decision tree that integrates multiple classification algorithms and leverages the strengths of each individual method. We also demonstrate that EnsBoD is able to identify bots in this dataset with higher accuracy and recall than any of the individual models alone.

In particular, we focus on the following research questions:
RQ1: How accurate are bot detection techniques?
RQ2: How prevalent are bots among the most active contributors?
RQ3: How active are bots in terms of commits?

Bot detection techniques. In this section, we evaluate the accuracy of the following five bot detection techniques:

1. *GitHub account type.* This technique relies on the GitHub API to determine whether a given GitHub account is a bot or not. The GitHub API offers an endpoint[6] to retrieve various metadata for a given GitHub username. Among other, these metadata includes a "type" field that is either "Bot" or "User" depending on whether the corresponding account had been registered as a bot or as a human contributor.

2. *"bot" suffix.* This technique relies on the presence of the string "bot" at the end of the author's name. It has been proposed by Dey et al. (Dey

[6]https://docs.github.com/en/rest/reference/users

et al., 2020b) as part of an ensemble model, and has notably been used by other researchers (Saadat et al., 2021).

3. *BoDeGHa.* In Section 5.3 we proposed a classification model to identify bots in GitHub pull request and issue activity. Their method measures the similarity of comments and groups them into patterns of similar comments. Bots are then detected based on their lower number of comments patterns. The model has been implemented as part of a tool named BoDeGHa.[7]

4. *BoDeGiC.* In Section 6.1.4 we further extended the above approach to support git commit messages, and implemented the resulting model as part of a tool named BoDeGiC.[8]

5. *List of bots.* This last technique relies on a predefined list of bots. The list contains the names of 527 known GitHub bot accounts that we manually identified among 5,000 GitHub accounts (explained in Chapter 4).[9]

6.4.1 Data extraction

To carry out our empirical investigation, we selected projects from active software development repositories with a large number of commits and contributors. We relied on the SEART GitHub search tool (Dabic et al., 2021) to filter a set of repositories. We queried repositories that have at least 100 contributors and were not forked and had been active in the last 2 months (i.e., in October and December 2021). From these, we randomly selected 27 large and active projects. The selected projects have at least 1,200 commits and 200 contributors. In total, the 27 selected projects account for 175,499 commits from 9,426 contributors and cover a wide variety of programming languages (e.g., Javascript, Python, Java, PHP, Ruby, Rust, Go) and software domains such as software development packages, plugins, and tools.

For each project, we queried the GitHub API to retrieve the 20 most active GitHub accounts in terms of commits, and their respective number of commits. The resulting dataset consists of 540 accounts. Since one of our goals is to evaluate the accuracy of bot detection techniques, we need to determine the correct type (i.e., bot or human) of these accounts. We manually checked these accounts to determine their type, looking for evidence in their profile, their commit activity and their commenting activity. During this process, we found 50 bots out of the 540 considered accounts.

[7]`https://github.com/mehdigolzadeh/BoDeGHa`
[8]`https://github.com/mehdigolzadeh/BoDeGiC`
[9]https://doi.org/10.5281/zenodo.4000388

6.4.2 How accurate are bot detection techniques?

We applied the five bot detection techniques on our dataset of 540 contributors. Fig. 6.5 at the end of this chapter, shows the classifications provided by these techniques. For readability, we only report on the 87 contributors that either correspond to actual bots, or that were classified as bot by at least one of the techniques. Actual bots are shown on the left side of the vertical blue line while actual human contributors are shown on its right. An orange cell indicates that the contributor was identified as a bot by the corresponding technique, while a blue cell indicates that it was identified as a human contributor. Grey cells correspond to cases where there is not enough information for the technique to determine the account type. In the case of BoDeGHa, this corresponds to contributors with less than 10 comments in pull requests or issues. In the case of BoDeGiC, this corresponds to contributors having less than 10 commits made with a committer name matching their GitHub account name.

From this figure, we observe that *list of bots*, *"bot" suffix* and *GitHub account type* are safer techniques, in the sense they do not wrongly classify human contributors as bots. At the same time, they missed many actual bots: from 19 for *list of bots* to 32 for *GitHub account type*. We also observe that *BoDeGiC* effectively captures most bots, but at the same time, wrongly considers several human contributors as bots. *BoDeGHa* exhibits a similar behaviour: it is able to capture 25 out of 50 bots, but wrongly classifies much more humans as bots than *BoDeGiC* (30 versus 9). We note that none of the techniques is perfectly effective in detecting bots. Except for a few cases, the five techniques do not even agree on whether a given account is a bot or not. However, only 4 of the actual bots are not detected as such by any of the techniques, suggesting that a combination of the techniques could lead to an improved bot detection model.

Table 6.11 reports on the precision, recall and F1-score of the aforementioned techniques applied on the whole dataset of 540 contributors, distinguishing these scores between bot and human contributors. For completeness, we also report on the overall weighted scores. Given there are far more human contributors than bot contributors in the dataset, these high scores (between 0.898 and 0.966) are mostly driven by the scores obtained for human contributors. To ease the interpretation of these scores, we also provide the scores for a ZeroR model classifying all contributors as human contributors (i.e., the majority class).

The observations that can be made from this table match the ones we made from Fig. 6.5. In particular, we observe that some techniques (namely *GitHub* *account type*, *"bot" suffix* and *list of bots*) have a perfect precision but are not able to capture as many bots as *BoDeGiC*. This should not come as a surprise. For example, it is expected that *GitHub account type* has no false positive since it is unlikely that a human contributor would decide to flag his/her own account as a bot. Similarly, *list of bots* relies on a predefined list of bot names that were manually validated by a group of researchers. On the other hand, the precision reached by *"bot" suffix* is surprisingly high since in Chapter 4, we found that only around 4% of the contributors having "bot" in their name actually correspond to human contributors.

As observed from Fig. 6.5, only four of the actual bots are not detected as such by any of the techniques. This suggests that an improved bot detection model can be created by combining the five aforementioned techniques. We build such a model by training a decision tree classifier taking as input the classifications made by each of the five techniques and outputting whether the corresponding contributor is a bot or a human contributor. Since our dataset has a fairly imbalanced number of human and bot contributors, we attributed a class weight inversely proportional to the number of cases. The resulting model is called EnsBoD. We trained and validated it following a 10-fold cross-validation process. The mean scores we obtained are reported on the last row of Table 6.11. Even if it was trained and validated on a small dataset, the EnsBoD model already outperforms any of the five other techniques, with an average recall of 0.9 and an average precision of 0.865 for bots. In the remaining of this section, we will rely on EnsBoD to separate bots that are correctly identified as bots by a bot detection technique and those that were not, providing an overly optimistic view of the ability to detect bots automatically.

6.4.3 How prevalent are bots among active contributors?

In Section 3.2 we underlined the importance of detecting bots in software repositories, not only for researchers aiming at quantifying and understanding their impact on the development process, but also for properly recognizing and rewarding contributions made by human contributors. This question aims to quantify the prevalence of bots among the 20 most active contributors in the 27 considered projects.

We applied EnsBoD on each of the 540 contributors of our dataset to quantify how many of them can be captured by the bot detection technique. Fig. 6.6 shows the output of EnsBoD for each project (x-axis) and each contributor (y-axis) sorted by the number of commits they made in the project. In complement to the output of EnsBoD (i.e., "bot" or "human"), we indicate whether the output is correct ("human user" and "correctly classified bot") or not ("human classified as bot" and "missed bot").

We observe that all the considered projects are making use of bots, some of them even having 4 different bots among their 20 most active contributors. Interestingly, many of these bots are responsible for most of the activity in the projects. For instance, the most active contributor of 6 projects is a bot, while 18 out of 27 projects have a bot in the top 3 contributors.

We also observe that a non-negligible amount of bots are missed even by our overly optimistic EnsBoD model. For instance, 5 bots are missed and 3 of them are among the 5 most active contributors of the projects. Similarly, a non-negligible amount of actual human contributors are wrongly classified by EnsBoD: there are 7 human contributors that are detected as bots, of which 1 is the most active contributor in the corresponding project, and 5 others are within the 10 most active contributors.

These findings show the importance of considering bots and their activity in software repositories, not only for conducting empirical research but also when acknowledging or rewarding contributors. While bot detection techniques can help in doing so, even an optimistic combination of them still misses some bots,

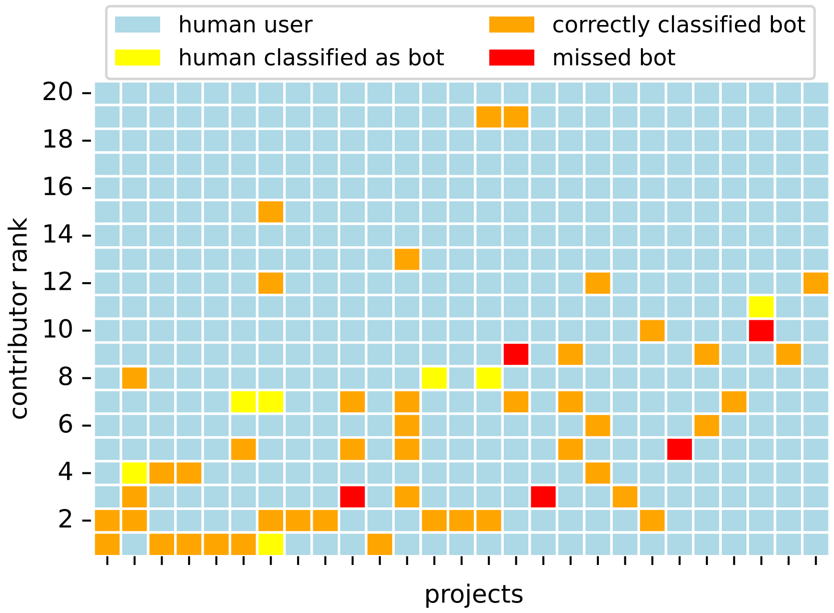


Figure 6.6: Rank of top 20 most active contributors in 27 popular open-source software projects.

and still wrongly considers some human contributors as bots.

6.4.4 How active are bots in terms of commits?

This question aims to quantify the number of commits made by bots in their respective projects. This is especially important given that tools such as *SourceCred* reward contributors based on their activity, including their commit activity. For each project, we counted the commits made by each of the 20 most active contributors, distinguishing between bot and human contributors. Fig. 6.7 reports on the proportion of commits made in each project. As for Fig. 6.6, we distinguish between human contributors, human contributors classified as bots, bot contributors and bot contributors missed by EnsBoD.

The figure shows that the commits made by bots represent up to 69.7% of the commit activity in one of the projects. On average, approximately 16% of the commits in these projects are made by bots (median is 12%), even if bots only account for 9% of the top 20 contributors on average (median is 10%)

While, as observed in previous research question, EnsBoD is able to detect most of the bots, it still misses some of them, and the missed ones are responsible for 8%, 7.3%, 4.2%, 2.5% and 1.7% of the commits in their respective projects (i.e., 4.7% on average). On the other hand, EnsBoD wrongly classified seven human contributors as bots, and these contributors were responsible for 38.4%, 11.5%, 4.8%, 1.5% and 1.2% of the commits (i.e., 10.4% on average).

This again underlines the importance of considering bots when analysing commit activity in software repositories, and highlights the need for better bot detection techniques to do so.

6.5. Summary and conclusions

This chapter reported on how we extended the bot identification technique of Chapter 5 in different ways and directions in order to mitigate its limitations. In Section 6.1, we introduced a different classification model and corresponding tool based on commit messages. This addresses the limitation of the model of Chapter 5 that is only based on pull request and issue comments.

As another limitation, the initial model relied on the fact that bots produce similar messages, therefore limiting us in situations where the bots did not (yet) provide a sufficient number of comments. To address this limitation, in Section 6.2 we explored a different technique based on the content of the messages and Natural Language Processing. This more fine-grained classification model

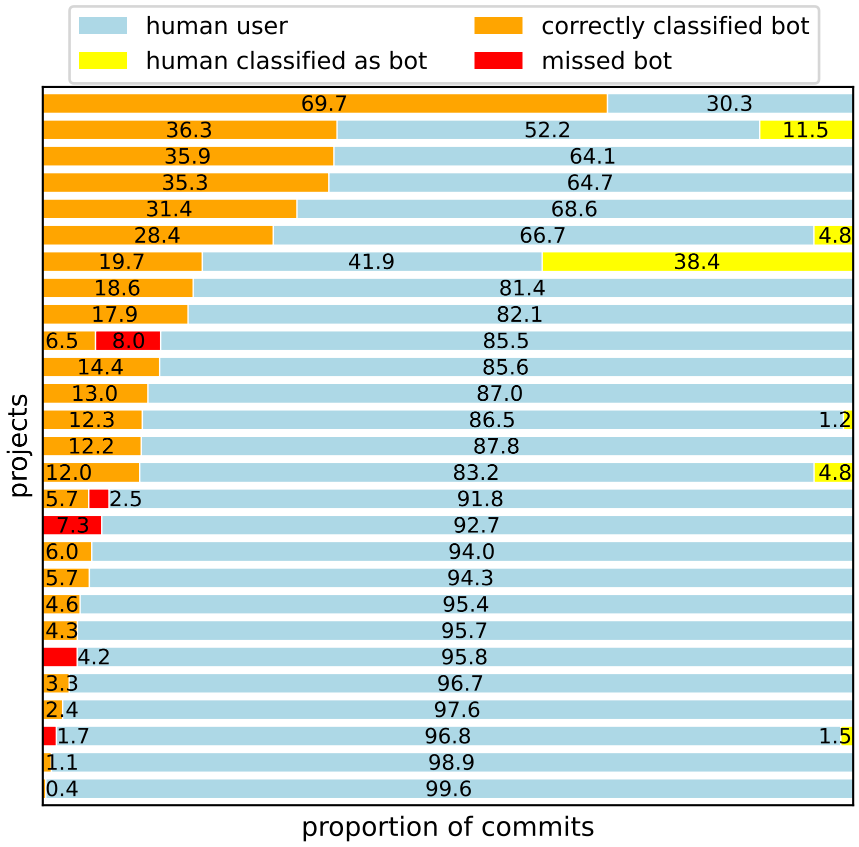


Figure 6.7: Proportion of commits made by the 20 most active contributors in each project

detects bots at the level of individual activities rather than at the level of the entire account. The model also opens the door to identify mixed accounts in GitHub. We also addressed in Section 6.3 the lack of sufficient comments by considering that accounts can be active in multiple repositories. Therefore, we introduced a technique to improve the results using divergent predictions from multiple repositories. Less active accounts are more likely to be predicted correctly using this technique.

Finally in Section 6.4, we compared existing bot identification techniques based on a small dataset of repositories. This comparison revealed that none of the techniques can be used to identify and cover all the existing types of bots. Therefore, we developed an ensemble model that incorporates all these techniques. Although most of the bots have been correctly identified by the ensemble model, there are still a few that could not be identified. Our bot identification technique has been utilized in various studies, serving as a valuable tool or dataset for identifying and studying bots. Researchers have used it to analyze and remove bots from datasets, ensuring unbiased results in their investigations.

Table 6.11: Recall, precision and $F1$-score of bot detection techniques (in ascending order of bot recall).

	bots			**humans**			**overall (weighted scores)**		
bot detection technique	recall	precision	$F1$-score	recall	precision	$F1$-score	recall	precision	$F1$-score
Baseline ZeroR	0.000	0.000	0.000	1.000	0.907	0.951	0.907	0.823	0.863
GitHub account type	0.360	**1.000**	0.529	**1.000**	0.939	0.968	0.941	0.944	0.928
BoDeGHa	0.500	0.455	0.476	0.939	0.948	0.944	0.898	0.903	0.900
“bot” suffix	0.520	**1.000**	0.684	**1.000**	0.953	0.976	0.956	0.958	0.949
List of bots	0.620	**1.000**	0.765	**1.000**	0.963	0.981	0.965	0.966	0.961
BoDeGiC	0.680	0.791	0.731	0.982	0.968	0.975	0.954	0.951	0.952
EnsBoD	**0.900**	0.865	**0.882**	0.986	**0.990**	**0.988**	**0.978**	**0.978**	**0.978**

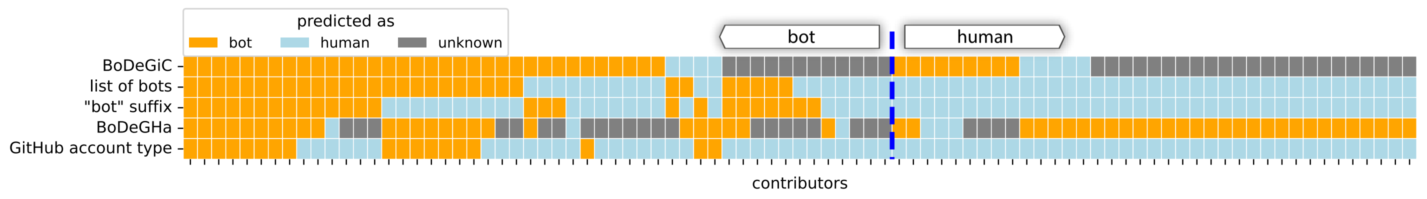


Figure 6.5: Classifications (“bot”, “human” or “unknown”) obtained from the five bot detection techniques. Only actual bots and humans wrongly classified as bot are displayed.

CHAPTER 7

Conclusions

“"The true worth of a man is to be measured by the objects he pursues.”

Avicenna

The advent of social coding platforms and accordingly workflow automation tools has significantly changed the way developers contribute to software projects, especially in open-source development. As we described in detail in Chapter 2, many studies have been conducted in order to better understand the social coding platforms and development automation tools such as CIs and bots. Studies in this area provide models, frameworks and tools and investigate different aspects of software development automation tools. However, further research is needed as there are still many unknowns.

Existing methods to detect automation tools in social coding platforms often relied on simple heuristics or manual inspection. We therefore explored the evolution of such tools and proposed techniques to detect them in social and technical aspect of social coding platforms. The present dissertation provided evidence and support for the following thesis statement:

Thesis statement

Development bots are increasingly used to automate more and more aspects of collaborative open-source software development. This raises the need to accurately identify bots and their automated activities in social coding platforms.

This concluding chapter summarizes our goals and contributions and explains how they contribute to the body of software engineering knowledge and practice. We discuss the limitations and shortcomings of the proposed techniques and tools and the provided solutions. Finally, we shed light on how these bot identification techniques can be exploited in empirical research, we discuss the perspectives we envision for the next generation of development automation detection techniques and how the research presented in this dissertation could be continued.

7.1. Contributions

This dissertation presented a series of studies that were conducted throughout my PhD. It contains empirical studies, developed classification models and tools that are described from Chapter 3 to Chapter 6. This section summarizes the contributions, explaining how each contribution supports the thesis statement. Section 7.3 discusses how others researchers can benefit from the contributions as well as future perspectives.

7.1.1 Created datasets

In order to study CI usage, we created a dataset of GitHub development repositories of npm packages containing 201,403 active repositories and we identified evidence of the presence of CI tool usage in those repositories. This dataset includes longitudinal usage histories of 20 different CI tools in these repositories. We used this dataset to investigate the usage, migration and co-usage of CI tools in the npm ecosystem.

In order to train and evaluate a bot classification model we created a ground-truth dataset of human and bot accounts. To create this ground-truth dataset, we downloaded issue and pull request commenting activities from a large number of GitHub accounts. We designed and developed a rating application that allowed us to classify 5,000 GitHub accounts. The resulting dataset consists of 527 bot accounts and 4,473 human accounts. The dataset constitutes an important contribution of our work since at the time we started our research there was no such dataset of GitHub bots available.

7.1.2 Classification models to identify bot accounts

The ground-truth dataset that we created allowed us to characterize bots based on their commenting activity in GitHub issue and pull requests. We measured the pairwise similarity of comments using two well-known similarity metrics, Jaccard and Levenshtein. We created clusters of comments using the DBSCAN clustering algorithm to find the number of comment patterns of an account. To be able to compare the inequality in comment distribution in comment patterns we used the Gini inequality metric. In total, we found four distinguishing features to separate bot accounts from human accounts including: the number of comment patterns, the number of comments, the number of empty comments and the distribution of comments within patterns.

The identified features along with the ground truth dataset of human and bot accounts allowed us to train a classification model to identify bots. To be able to select an appropriate classifier we compared five well-known binary classifiers including *random forest classifier*, *support vector machine*, *logistic regression*, *k-nearest neighbours* and *decision trees.* The random forest classifier showed the best performance and was chosen as the final model. This classification model was evaluated based on the remaining 40% of the dataset. Our classification model was able to detect most bots as well as human accounts. However, it had some false positives and false negatives that we examined carefully to recategorize the types of accounts that were incorrectly classified.

The classification model we developed was only able to identify bots active in pull requests and issue comments. However, bots may also be active in other aspects of the software development process such as code contribution. To address this shortcoming we developed a new classification model to identify bots in commit activities. We used the same features to detect bots using commit messages. To train and evaluate the new classification model we created a dataset of commits associated with a set of repositories and their corresponding commit messages. Comparing the five binary classifiers, we found that random forest was the most effective.

In view of the presence of mixed accounts, we also created another classification model to identify bots not at the level of an account but at the level of their individual activities. In other words, the question “Is this account a bot?” would become "Is this specific activity produced by a bot?”. Based on NLP techniques, we generated a probabilistic classification model to identify the type of activities instead of the type of accounts. The new classification model also provides a probabilistic prediction that makes it possible to identify accounts that are shared between a human contributor and a bot (Mixed accounts).

Another contribution of this dissertation is a classification model based on multiple repositories. To improve the predictions of our classifier we used predictions of our bot identification tool provided for a single account that is active across multiple repositories. We created a dataset of accounts from development repositories associated with the Cargo ecosystem to investigate the accuracy of this classification model. The resulting model called WoC-P is relying on the wisdom of the crowd. WoC-P improved the bot detection accuracy in the considered set of accounts. This suggests that relying on the wisdom of the crowd principle is a promising approach to improve bot detection models across repositories. Additionally, we demonstrated that incomplete predictions, where the model failed to predict an account type, can be rectified by combining predictions from multiple repositories.

The last contribution of this dissertation is a comparison between five existing bot detection techniques. We conducted an exploratory study on the accuracy of five bot detection techniques on a dataset containing the top 20 most active contributors of a set of 27 large projects. The result of the analysis revealed that none of the compared techniques is perfectly effective in detecting bots. For this reason, we combined these five bot detection techniques in an ensemble classifier, called EnsBoD, that incorporates the capability of all individual techniques in order to improve the bot detection accuracy. We evaluated EnsBoD through a 10-fold cross-validation process, and we found that EnsBoD exhibits more accurate predictions.

7.1.3 Bot detection tools

Another contribution of this dissertation are the tools that we developed and published based on the above-mentioned classifiers. We implemented the classification model that we developed for bot identification in GitHub pull request and issues, into a Python command-line tool called BoDeGHa. This open-source tool is made freely available to allow practitioners and researchers to analyse GitHub repositories and to identify which accounts correspond to bots and which correspond to humans. The tool makes the classification model easily available and usable. BoDeGHa downloads the required information from the GitHub API and predicts the type of accounts and produces the output in different formats. The tool is available on GitHub at the following address:
`https://github.com/mehdigolzadeh/BoDeGHa`

We also used the classification model we built to identify bots in commit activities to develop a second bot identification tool. The open-source command-line tool is called BoDeGiC. It requires as input a git repository and extracts commit information to classify human and bot authors in the repository. The tool is available on GitHub at the following address:
`https://github.com/mehdigolzadeh/BoDeGiC`

7.2. The Impact of the Contributions

In this section, we delve into the impact of the contributions presented in Section 7.1 by examining cases where our bot identification technique, bot identification tools (BoDeGHa and BoDeGiC), and the ground-truth dataset of bots have been utilized for bot identification purposes. These studies not only validate the effectiveness of our approaches but also demonstrate the wider applicability and usefulness of the techniques, tools, and datasets developed throughout this research. By analyzing the adoption and integration of these resources in various third-party studies, we gain insights in the influence and significance of our contributions in the field of bot identification and its implications for collaborative software development on social coding platforms such as GitHub.

In the past few years, several studies have been conducted to explore the different aspects of GitHub and collaborative software development. Studies focusing on socio-technical analysis have eliminated bot activities (Foundjem et al., 2022) in order to avoid wrong conclusions. Among these studies, we analysed cases where either our bot identification technique, bot identification tool, or the ground-truth dataset of bots were used for bot identification purposes (Gao et al., 2022; Abdellatif et al., 2022; Cassee et al., 2021).

Abdellatif et al. (2022) developed a technique to identify bots that are active in both commit and issue activities. They have compared our bot identification technique to with theirs. Cassee et al. (2021) used our method of identifying bots (presented in Chapter 6) as one of the techniques for identifying so-called mixed accounts in GitHub. Their methodology for classifying GitHub pull requests and issue comments consists of three different methods, and they use two of the techniques described in Chapter 4 and 6 for bot identification.

In addition to studies focused on introducing a new bot identification method, there are also studies that utilize our tools and techniques to identify bot activities. Some of these studies have used our publicly available ground-truth dataset of bot accounts in order to identify bot activities. Schueller et al. (2022) have compiled a dataset of collaboration and dependency repository maintainers within the *rust* open-source programming ecosystem. In order to clean bot activities from GitHub artifacts they used our ground-truth dataset of bot accounts (presented in Chapter 4). This dataset is also used in other studies to create a methodological framework to measure risk in software ecosystems (Schueller & Wachs, 2022) and to understand how software engineers react to bots' actions (Cassee, 2022).

Furthermore, some studies have used our bot identification tool BoDeGHa to identify bots. Some researchers examined the impact of the presence of bots. For example, Zhang et al. (2022b) have investigated the effect of the presence of bots in pull request evaluation, Wang et al. (2022) identified how open-source software projects adopt bot services from a diverse set of available bots and Wu et al. (2022) used our tool to automatically identify bots and characterize the current usage of bots in practices.

Other studies ave removed bot activities to avoid potential biases of having bots in their datasets. For instance, in the study conducted by Joblin et al. (2022) about the organizational structure of open-source projects, BoDeGHa was used to remove bot comments where the authors intended to create a developer network through comments. Enache et al. (2021) evaluated the impact of adopting a "code of conduct" on women's participation in open-source communities in GitHub. To do so they used BoDeGHa to remove bot accounts from their dataset.

7.3. Future research perspectives

7.3.1 Bot identificaton and dataset improvement

While the bot classification models presented in this dissertation show a promising result in identifying bots, there is still room for improvement. For example, researchers can explore to what extent it is possible to go beyond comments and incorporate temporal information, account specifications and other technical information, type of activities and in general other types of metadata in GitHub repositories to identify bots. In fact, this is what already happening. Recently Abdellatif et al. (2022) have proposed a classification model that extends our model with additional information from account specifications such as account name, account bio, number of followings and followers and account activity such as total number of activities in a repository, number of pull requests and issues, median activity per day and median response time.

Moreover, extracting additional types of information from GitHub, such as code review activities and GitHub discussions, can provide valuable insights for bot identification. Factors like the frequency of activity, the number of activity types, and engagement in multiple repositories can be considered as features in a classification model. Additionally, analyzing the temporal aspect such as time between consecutive activities, which is currently being explored by one of the researchers in our lab (Chidambaram et al., 2023a,b,c), can further contribute to distinguishing between bot and human accounts. By incorporating these additional features, the classification model can gain a more comprehensive understanding of bot behavior and improve its ability to differentiate between bot accounts and human accounts.

In addition to the data available on GitHub, researchers can explore the utilization of data from other software engineering resources. Software developers rely on various issue tracking, code revewing and bug tracking tools like Gerrit, JIRA, and Bugzilla throughout the development process, each with its own unique data sources. These tools can provide valuable information for identifying bot activity. Integrating bot activity data from these different tools into existing bot identification models can offer a more comprehensive understanding of bot behavior. This integration can also enhance the accuracy of bot detection models, as they would be able to capture a broader range of bot behaviors. As bots continue to evolve and become more sophisticated, it is crucial to collect data from diverse sources to ensure that bot detection models remain effective. By considering data from various software engineering resources, researchers can stay up-to-date with the latest bot behaviors and adapt their models accordingly. This multi-source approach will contribute to a more comprehensive and accurate understanding of bot activity in software development.

Expanding the scope of bot identification research beyond GitHub to other social coding platforms such as GitLab and Bitbucket presents an intriguing avenue for future investigation. While this dissertation has primarily focused on bot identification in GitHub, it is essential to recognize that bots are active across various social coding platforms, each potentially exhibiting unique behaviors. Therefore, developing techniques tailored to identify bots in these platforms would contribute to a more comprehensive understanding of software development bots and foster improved bot detection mechanisms across the social coding ecosystem.

In this dissertation, we also presented a classification model based on diverging predictions of BoDeGHa. Although the model shows promising results, it has not been integrated into the BoDeGHa tool itself. Integrating the model into BoDeGHa would be an interesting next step, as it could eliminate the limitations faced by BoDeGHa in predicting accounts with few comments. Being a flexible open-source tool, Bodegha has the potential to incorporate more advanced models, as discussed earlier by considering additional data sources, activity types, and characteristics. Furthermore, it can compute a broader range of features to enhance its bot identification capabilities. Integrating the new classification model into BoDeGHa would further increase the value of the tool for both researchers and practitioners.

Although we trained our models on a dataset comprising both bot and human accounts, the rapidly-evolving nature of bots means that our model may not be able to accurately predict bots in the future. Therefore, having a classification model that can retrain itself incrementally and adapt to the new set of bots would be a very interesting future perspective. This becomes increasingly important as we observe that bot developers use more advanced machine learning techniques, such as GPT, to create software development bots. Such powerful language models have demonstrated their ability to generate human-like text and engage in conversations that are difficult to distinguish from those of human users. As these sophisticated bots generate more human-like comments, they become less distinguishable for the classification models that rely solely on comment patterns. The traditional approach of analyzing comment content may be less effective in identifying these advanced bots. To tackle this challenge effectively, integrating language models capable of distinguishing text generated by such models into our classification framework could enhance the accuracy of bot identification. By leveraging the capabilities of advanced language models, we can develop more robust algorithms that consider not only comment patterns but also the semantic meaning and contextual understanding of the text. Continuous retraining and improvement of the classification model with a diverse range of bot behaviors, including those exhibited by models like GPT, would be crucial to stay ahead of emerging bot developments and ensure effective bot detection in the future.

Aside from being used by other studies and by practitioners, the datasets can be extended with more information and continuously updated. Given that the process of rating the accounts and labeling them was very time-consuming, we limited ourselves to 5,000 accounts. Considering that we detected these bots in a limited set of repositories, we are confident that thousands of others could be added to the dataset. In addition, more and more use-cases are defined for bots in social coding platforms, and so new bots will definitely be developed to handle these tasks. Therefore, updating the dataset with emerging bots would be definitely beneficial to have a more complete list of active bots in GitHub. It would be interesting to include the types of GitHub activities of each account and also mixed accounts in the dataset. One can go beyond GitHub and add bots that are active in other platforms (likewise CIs active in other platforms to CI dataset).

The study by Erlenhov et al. (Erlenhov et al., 2020b) revealed that a more fine-grained characterization of development bots would be needed. They came up with three DevBot *personas* based on their autonomy, chat interface, and smartness. Such a more fine-grained classification could be used to refine our ground-truth dataset. This dataset can be used for other study purposes, including the identification of bots in a dataset related to the study of the history of GitHub repositories. Researchers can also use this collection to develop better models for account classification.

7.3.2 Enhance software development processes

Another research track enabled by bot classification models is the empirical investigation of the impact of bots on software development processes. Accurately identifying bots allows researchers to delve into how bot activities influence the collaborative software development process. This opens up opportunities to examine the specific ways in which bots contribute to, and improve various aspects of software development, such as project management, dependency management, bug tracking, and code review. By studying the engagement of bots in these domains, researchers can acquire valuable insights into the efficiency, effectiveness, and potential challenges associated with bot involvement (Rombaut et al., 2023). For instance, they can analyze how bots streamline project management tasks by automating certain actions or how they facilitate bug tracking through automated testing and reporting. Additionally, researchers can explore the impact of bots on code review processes, investigating their role in identifying issues, providing feedback, and improving code quality.

Examining the social dynamics of software development in the presence of bots is another valuable avenue for research. Understanding how bots interact with human contributors and how they affect collaboration and communication is essential for optimizing team dynamics. Researchers can investigate the ways in which bots facilitate or hinder collaboration among developers, as well as the implications for knowledge sharing, decision-making, and overall productivity. Leveraging bot classification models researchers can conduct empirical studies to uncover the multifaceted impact of bots on software development processes. This research not only enhances our understanding of the role of bots but also provides insights into how to effectively integrate and leverage bot capabilities to drive innovation in collaborative software development.

7.3.3 Bot behavior optimization

Bot behavior optimization is a compelling research area within collaborative software development. Researchers can explore how to design bots that work effectively in different types of projects and prevent their negative impacts on the development process. This involves customizing bot behavior based on project requirements and specific workflows. By tailoring bot capabilities and functionalities to align with project objectives, researchers can develop more efficient bots. They can also investigate methods to ensure bots comply with ethical and legal standards, preventing disruptions and conflicts within the social codding environment. Establishing mechanisms for monitoring and regulating bot activities can foster a harmonious collaboration between bots and human contributors.

The interactions between bots and human contributors present another intriguing area for exploration. Researchers can optimize collaboration and communication between bots and humans, aiming for seamless integration and streamlined knowledge exchange. This involves studying the dynamics of bot-human interactions, identifying challenges, and proposing strategies to enhance collaboration and teamwork. By understanding the strengths and limitations of bot-human partnerships, researchers can develop guidelines and best practices to foster a productive and supportive working environment. The findings from this research will ultimately contribute to the development of more efficient, ethical, and productive software development processes, while simultaneously advancing collaborative practices through the continued improvement and refinement of bot capabilities.

Bibliography

Aattouri, I., Rida, M., & Mouncif, H. (2021). Creation of a callbot module for automatic processing of a customer service calls.

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021). A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering*, (pp. 1–1).

Abdellatif, A., Badran, K., & Shihab, E. (2020). Msrbot using bots to answer questions from software repositories. *Empirical Software Engineering*.

Abdellatif, A., Wessel, M., Steinmacher, I., Gerosa, M. A., & Shihab, E. (2022). BotHunter: An approach to detect software bots in GitHub.

Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*, *2*, 100006. URL `https://www.sciencedirect.com/science/article/pii/S2666827020300062`

Adikari, A., de Silva, D., Moraliyage, H., Alahakoon, D., Wong, J., Gancarz, M., Chackochan, S., Park, B., Heo, R., & Leung, Y. (2022). Empathic conversational agents for real-time monitoring and co-facilitation of patient-centered healthcare. *Future Generation Computer Systems*, *126*, 318–329. URL `https://www.sciencedirect.com/science/article/pii/S0167739X2100323X`

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Ahlgren, J., Berezin, M., Bojarczuk, K., Dulskyte, E., Dvortsova, I., George, J., Gucevska, N., Harman, M., Lomeli, M., Meijer, E., Sapora, S., & Spahr-

Summers, J. (2021). Testing web enabled simulation at scale using metamorphic testing. In *IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, (pp. 140–149).

Akma, N., Hafiz, M., Zainal, A., Fairuz, M., Adnan, Z., & Adnan, Z. (2018). Review of chatbots design techniques. *International Journal of Computer Applications*.

Aldayel, A., & Magdy, W. (2022). Characterizing the role of bots' in polarized stance on social media. *Social Network Analysis and Mining*.

Alizadeh, V., Ouali, M. A., Kessentini, M., & Chater, M. (2019). RefBot: Intelligent software refactoring bot. In *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (pp. 823–834).

Allameh, M., & Zaman, L. (2021). Jessy: A conversational assistant for tutoring digital board games. In *Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '21, (p. 168–173). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3450337.3483496`

Amleshwaram, A. A., Reddy, N., Yadav, S., Gu, G., & Yang, C. (2013). CATS: Characterizing automation of Twitter spammers. *International Conference on Communication Systems and Networks (COMSNETS)*.

Amrit, C., & Meijberg, Y. (2018). Effectiveness of test-driven development and Continuous Integration: A case study. *IT Professional*, *20*(1), 27–35.

Arciniegas-Mendez, M., Zagalsky, A., Storey, M.-A., & Hadwin, A. F. (2017). Using the model of regulation to understand software development collaboration practices and tool support. In *ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, (p. 1049–1065). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2998181.2998360`

Arora, R., Goel, S., & Mittal, R. (2016). Supporting collaborative software development over GitHub. *Software: Practice and Experience*, *47*.

Bagmar, A., Hogan, K., Shalaby, D., & Purtilo, J. (2022). Analyzing the effectiveness of an extensible virtual moderator. *Proc. ACM Hum.-Comput. Interact.*, *6*(GROUP).
URL `https://doi.org/10.1145/3492837`

Balachandran, V. (2013). Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *International Conference on Software Engineering (ICSE)*, (pp. 931–940). IEEE.

Baltes, S., Knack, J., Anastasiou, D., Tymann, R., & Diehl, S. (2018). (no) influence of Continuous Integration on the commit activity in GitHub projects. In *4th ACM SIGSOFT International Workshop on Software Analytics*, SWAN 2018, (p. 1–7). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3278142.3278143`

Baudry, B., Chen, Z., Etemadi, K., Fu, H., Ginelli, D., Kommrusch, S., Martinez, M., Monperrus, M., Ron, J., Ye, H., & Yu, Z. (2021). A software-repair robot based on continual learning. *IEEE Software*, *38*(4), 28–35.

Beck, K., & Andres, C. (2004). Extreme programming explained: Embrace change (2nd edition).

Beller, M., Gousios, G., & Zaidman, A. (2017). Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *International Conference on Mining Software Repositories (MSR)*, (pp. 356–367).

Ben Mimoun, M. S., Poncin, I., & Garnier, M. (2017). Animated conversational agents and e-consumer productivity: The roles of agents and individual characteristics. *Information & Management*, *54*(5), 545–559.

Bernardo, J. H., da Costa, D. A., & Kulesza, U. (2018). Studying the impact of adopting continuous integration on the delivery time of pull requests. In *IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, (pp. 131–141). IEEE.

Beschastnikh, I., Lungu, M. F., & Zhuang, Y. (2017). Accelerating software engineering research adoption with analysis bots. *International Conference on Software Engineering: New Ideas and Emerging Results Track*, (pp. 35–38).

Bird, C. (2011). Socio-technical coordination and collaboration in open source software. In *27th IEEE International Conference on Software Maintenance (ICSM)*, (pp. 568–573).

Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D., German, D. M., & Devanbu, P. (2009). The promises and perils of mining git. In *Working Conference*

on Mining Software Repositories (MSR). IEEE Computer Society.
URL `https://www.microsoft.com/en-us/research/publication/the-promises-and-perils-of-mining-git/`

Bisong, E., Tran, E., & Baysal, O. (2017). Built to last or built too fast? Evaluating prediction models for build times. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 487–490).

Bjorn, P., Bardram, J., Avram, G., Bannon, L., Boden, A., Redmiles, D., de Souza, C., & Wulf, V. (2014). Global software development in a CSCW perspective. In *17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW Companion '14, (p. 301–304). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2556420.2558863`

Black, D., Rapos, E. J., & Stephan, M. (2019). Voice-driven modeling: software modeling using automated speech recognition.

Bradley, N. C., Fritz, T., & Holmes, R. (2018). Context-aware conversational developer assistants. In *40th International Conference on Software Engineering*, ICSE '18, (p. 993–1003). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3180155.3180238`

Brandtzæg, P. B., & Følstad, A. (2017). Why people use chatbots.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Brown, C. (2019). Digital nudges for encouraging developer actions. In *IEEE/ACM 41st International Conference on Software Engineering*, (pp. 202–205).

Brown, C., & Parnin, C. (2019). Sorry to Bother You: Designing bots for effective recommendations. *1st International Workshop on Bots in Software Engineering*, (pp. 54–58).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Buchanan, S., Rangama, J., & Bellavance, N. (2020). *CI/CD with Azure Kubernetes Service*, (pp. 191–219). Berkeley, CA: Apress.
URL `https://doi.org/10.1007/978-1-4842-5519-3_9`

Burridge, J. (1991). Statistics in society. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, *154*(2), 361–364.
URL `http://www.jstor.org/stable/2983054`

Campbell, J. L., Quincy, C., Osserman, J., & Pedersen, O. K. (2013). Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research*, *42*(3), 294–320.

Cánovas Izquierdo, J. L., & Cabot, J. (2021). On the analysis of non-coding roles in open source development. *Empirical Software Engineering*, *27*(1).

Cantador, I., Viejo-Tardío, J., Cortés-Cediel, M. E., & Rodríguez Bolívar, M. P. (2021). A chatbot for searching and exploring open data: Implementation and evaluation in E-Government. In *DG.O2021: The 22nd Annual International Conference on Digital Government Research*, DG.O'21, (p. 168–179). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3463677.3463681`

Carvalho, A., Luz, W., Marcílio, D., Bonifácio, R., Pinto, G., & Dias Canedo, E. (2020). C-3PR: A bot for fixing static analysis violations via pull requests. In *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (pp. 161–171).

Cassee, N. (2022). Sentiment in software engineering: Detection and application. In *30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, (p. 1800–1804). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3540250.3558908`

Cassee, N., Kitsanelis, C., Constantinou, E., & Serebrenik, A. (2021). Human, bot or both? A study on the capabilities of classification models on mixed accounts. In *International Conference on Software Maintenance*. IEEE.

Cerezo, J., Kubelka, J., Robbes, R., & Bergel, A. (2019). Building an expert recommender chatbot. *IEEE/ACM 1st International Workshop on Bots in Software Engineering, BotSE 2019*, (Dcc), 59–63.

Chamberlain, W., & Hall, J. (1984). *Racter, The Policemanś Beard Is Half Constructed: Computer Prose and Poetry by RACTER*. Warner Books.

Chandrasekara, C., & Herath, P. (2021). *Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications*. Apress.

Chao, M.-H., Trappey, A. J. C., & Wu, C.-T. (2021). Emerging technologies of natural language-enabled chatbots: A review and trend forecast using intelligent ontology extraction and patent analytics. *Complexity*.

Chatterjee, P., Damevski, K., & Pollock, L. (2021). Automatic extraction of opinion-based Q&A from online developer chats. In *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, (pp. 1260–1272).

Chen, L. (2015). Continuous Delivery: Huge benefits, but challenges too. *IEEE Software*, *32*(2), 50–54.

Cheng, J., & Guo, J. L. C. (2019). Activity-based analysis of open source software contributors: Roles and dynamics. In *International Workshop on Cooperative and Human Aspects of Software Engineering*, (pp. 11–18). IEEE/ACM.

Chidambaram, N., Decan, A., & Golzadeh, M. (2022). Leveraging predictions from multiple repositories to improve bot detection. In *International Workshop on Bots in Software Engineering*.

Chidambaram, N., Decan, A., & Mens, T. (2023a). A dataset of bot and human activities in GitHub. In *International Conference on Mining Software Repositories (MSR)*. IEEE.

Chidambaram, N., Decan, A., & Mens, T. (2023b). A dataset of bot and human activities in GitHub.

Chidambaram, N., Decan, A., & Mens, T. (2023c). Distinguishing Bots from Human Developers Based on Their GitHub Activity Types. In *Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*. CEUR.

Cicco, R. D., Iacobucci, S., Aquino, A., Alparone, F. R., & Palumbo, R. (2022). Understanding users' acceptance of chatbots: An extended TAM approach. *Lecture Notes in Computer Science*.

Claps, G. G., Berntsson Svensson, R., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, *57*, 21–31.
URL `https://www.sciencedirect.com/science/article/pii/S0950584914001694`

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, *114*(3), 494–509.

Colace, F., Santo, M. D., Lombardi, M., Pascale, F., Pietrosanto, A., & Lemma, S. (2018). Chatbot for e-learning: A case of study.

Colby, K. M., Weber, S., & Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, *2*(1), 1–25.
URL `https://www.sciencedirect.com/science/article/pii/0004370271900026`

Collin-Sussmen, B., Fitzpatrick, B., & Pilato, C. (2002). *Version Control For Subversion For: Subversion 1.7. Version Control With Subversion*.

Constantino, K., Zhou, S., Souza, M., Figueiredo, E., & Kästner, C. (2020). Understanding collaborative software development: An interview study. In *15th International Conference on Global Software Engineering*, ICGSE '20, (p. 55–65). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3372787.3390442`

Cook, T. D., & Campbell, D. T. (1979). *Quasi-experimentation : design & analysis issues for field settings*. Boston: Houghton Mifflin.

Corti, G., Ambrosini, L., Guidi, R., & Rizzo, N. (2019). *Thing: Improve anything to anything collaboration. In *Advances in Information and Communication*, (pp. 453–465). Cham: Springer International Publishing.

Cosley, D., Frankowski, D., Terveen, L., & Riedl, J. (2007). SuggestBot: Using intelligent task routing to help people find work in Wikipedia. In *International Conference on Intelligent User Interfaces (IUI)*, (pp. 32–41). ACM.

Cruz, G. (2021). *Exploratory Graph Based Bot Detection Application on Reddit Subnetworks*. Ph.D. thesis, University of Maryland, College Park.

da Silva, B., Hebert, C., Rawka, A., & Sereesathien, S. (2020). Robin: A voice controlled virtual teammate for software developers and teams. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 789–791).

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. In *International Conference on Computer Supported Cooperative Work (CSCW)*, (pp. 1277–1286). ACM.

Dabic, O., Aghajani, E., & Bavota, G. (2021). Sampling projects in GitHub for MSR studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, (pp. 560–564). IEEE.

Dale, R. (2016). The return of the chatbots. *Natural Language Engineering*, *22*(5), 811–817.

Danglot, B., Monperrus, M., Rudametkin, W., & Baudry, B. (2019). An approach and benchmark to detect behavioral changes of commits in continuous integration. *arXiv: Software Engineering*.

Davis, A. M., Bersoff, E., & Comer, E. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*.

de Alwis, B., & Sillito, J. (2009). Why are software projects moving from centralized to decentralized version control systems? In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, (pp. 36–39).

De Jong, M., & Van Deursen, A. (2015). Continuous Deployment and schema evolution in SQL databases. In *IEEE/ACM 3rd International Workshop on Release Engineering*, (pp. 16–19).

Debbiche, A., Dienér, M., & Svensson, R. B. (2014). Challenges when adopting Continuous Integration: A case study. *PROFES*.

Debroy, V., Miller, S., & Brimble, L. (2018). Building lean Continuous Integration and Delivery pipelines by applying DevOps principles: A case study at Varidesk. In *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, (p. 851–856). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3236024.3275528`

Decan, A., Mens, T., Rostami Mazrae, P., & Golzadeh, M. (2022). On the use of GitHub actions in software development repositories. In *International Conference on Software Maintenance and Evolution (ICSME)*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics.
URL `https://aclanthology.org/N19-1423`

Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., & Mockus, A. (2020a). Detecting and characterizing bots that commit code. In *International Conference on Mining Software Repositories*, (pp. 209–219). ACM.

Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., & Mockus, A. (2020b). Detecting and characterizing bots that commit code. In *International Conference on Mining Software Repositories*, (pp. 209–219). ACM.
URL `https://doi.org/10.1145/3379597.3387478`

Dominic, J., Houser, J., Steinmacher, I., Ritter, C., & Rodeghero, P. (2020). Conversational bot for newcomers onboarding to open source projects. In *International Workhop on Bots in Software Engineering*, (pp. 46–50). New York, NY, USA: ACM.
URL `https://doi.org/10.1145/3387940.3391534`

Dorfman, R. (1979). A formula for the Gini coefficient. *The Review of Economics and Statistics*, *61*(1), 146–149.

Duvall, P., Matyas, S., Duvall, P., & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. A Martin Fowler signature book. Addison-Wesley.
URL `https://books.google.be/books?id=MA8QmAEACAAJ`

Ed-douibi, H., Daniel, G., & Cabot, J. (2020). OpenAPI bot: A chatbot to help you understand REST APIs.

Efthimion, P. G., Payne, S., & Nicholas, P. (2018). Supervised machine learning bot detection techniques to identify social Twitter bots. *SMU Data Science Review*, *1*(2).

Elazhary, O., Werner, C., Li, Z. S., Lowlind, D., Ernst, N. A., & Storey, M.-A. (2021). Uncovering the benefits and challenges of Continuous Integration practices. *IEEE Transactions on Software Engineering*.

Elkan, C. (2001). The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, (pp. 973–978). Morgan Kaufmann.

Embury, S. M., & Page, C. (2018). Effect of continuous integration on build health in undergraduate team projects. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, (pp. 169–183). Springer.

Enache, B., Constantinou, E., Serebrenik, A., & Kostitsyna, I. (2021). Effect of the introduction of code of conduct in collaborative development.

Erlenhov, L., de Oliveira Neto, F. G., & Leitner, P. (2022). Dependency management bots in open source systems—prevalence and adoption. *PeerJ*.

Erlenhov, L., Gomes de Oliveira Neto, F., Scandariato, R., & Leitner, P. (2019). Current and future bots in software development. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 7–11).

Erlenhov, L., Neto, F. G. d. O., Chukaleski, M., & Daknache, S. (2020a). Challenges and guidelines on designing test cases for test bots.

Erlenhov, L., Neto, F. G. d. O., & Leitner, P. (2020b). An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (pp. 445–455). ACM.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, (pp. 226—231). AAAI Press.

Farah, J. C., Spaenlehauer, B., Lu, X., Ingram, S., & Gillet, D. (2022). An exploratory study of reactions to bot comments on GitHub. (p. 5). New York: ACM.
URL `http://infoscience.epfl.ch/record/293085`

Farooq, U., & Grudin, J. (2016). Human-computer integration. *Interactions*, *23*(6), 26–32.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. ROC Analysis in Pattern Recognition.
URL `https://www.sciencedirect.com/science/article/pii/S016786550500303X`

Fergencs, T., & Meier, F. (2021). Engagement and usability of conversational search – a study of a medical resource center chatbot.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189.
URL `https://www.sciencedirect.com/science/article/pii/S0164121215001430`

Foundjem, A., Constantinou, E., Mens, T., & Adams, B. (2022). A mixed-methods analysis of micro-collaborative coding practices in OpenStack. *Empirical Software Engineering*, *27*(5), 1–55.

Fowler, M., & Foemmel, M. (2006). Continuous Integration.

Frank, E., & Hall, M. (2001). A simple approach to ordinal classification. In L. De Raedt, & P. Flach (Eds.) *European Confernece on Machine Learning (ECML)*, (pp. 145–156). Springer.

Følstad, A., Nordheim, C. B., & Bjørkli, C. A. (2018). What makes users trust a chatbot for customer service? An exploratory interview study. *Internet Science*, (pp. 194–208).

Gallaba, K., Lamothe, M., & McIntosh, S. (2022). Lessons from eight years of operational data from a Continuous Integration service: An exploratory case study of CircleCI. In *IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, (pp. 1330–1342).

Gallaba, K., & McIntosh, S. (2020). 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.

Gao, A., Chen, S., Wang, T., & Deng, J. (2022). Understanding the impact of bots on developers sentiment and project progress. In *IEEE 13th International Conference on Software Engineering and Service Science (ICSESS)*, (pp. 93–96).

Gautam, A., Vishwasrao, S., & Servant, F. (2017). An empirical study of activity, popularity, size, testing, and stability in Continuous Integration. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 495–498).

Geiger, R. S. (2009). The social roles of bots and assisted editing programs. In *5th International Symposium on Wikis and Open Collaboration*, WikiSym '09. New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/1641309.1641351`

Geiger, R. S. (2013). Are computers merely "supporting" cooperative work: Towards an ethnography of bot development. In *International Conference on Computer Supported Cooperative Work (CSCW)*, (pp. 51–56). ACM.

Geiger, R. S., & Halfaker, A. (2017). Operationalizing conflict and cooper-

ation between automated software agents in Wikipedia: A replication and expansion of 'Even Good Bots Fight'. *Proc. ACM Hum.-Comput. Interact.*, *1*(CSCW).
URL `https://doi.org/10.1145/3134684`

Ghaleb, T. A., Da Costa, D. A., & Zou, Y. (2019). An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, *24*(4), 2102–2139.

GitHub (2020). The 2020 state of the Octoverse - community report.
URL `octoverse.github.com`

GitHub (2021). The 2021 state of the octoverse - community report.
URL `octoverse.github.com`

Gkinko, L., Elbanna, A. R., & Elbanna, A. (2021). AI in the workplace: Exploring chatbot use and users' emotions.

Go, E., & Sundar, S. S. (2019). Humanizing chatbots: The effects of visual, identity and conversational cues on humanness perceptions. *Computers in Human Behavior*.

Goeminne, M., & Mens, T. (2013). A comparison of identity merge algorithms for software repositories. *Science of Computer Programming*, *78*(8), 971–986.

Golzadeh, M., Decan, A., & Chidambaram, N. (2022a). On the accuracy of bot identification tools. In *International Workshop on Bots in Software Engineering*.

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*.
URL `https://www.sciencedirect.com/science/article/pii/S016412122100008X`

Golzadeh, M., Legay, D., Decan, A., & Mens, T. (2020). Bot or not? Detecting bots in GitHub pull request activity based on comment similarity. *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 31–35).

Golzadeh, M., Mens, T., Decan, A., Constantinou, E., & Chidambaram, N. (2022b). Recognizing bot activity in collaborative software development. *IEEE Software*, *39*(5), 56–61.

Gousios, G., Pinzger, M., & Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *International Conference on Software Engineering (ICSE)*, (pp. 345–355). ACM.

Gousios, G., Storey, M.-A., & Bacchelli, A. (2016). Work practices and challenges in pull-based development: The contributor's perspective. In *International Conference on Software Engineering (ICSE)*, (pp. 285–296). ACM.

Gousios, G., Zaidman, A., Storey, M.-A., & Deursen, A. v. (2015). Work practices and challenges in pull-based development: The integrator's perspective. In *IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, (pp. 358–368).

Grbac, T. G., Mausa, G., & Basic, B. D. (2013). Stability of software defect prediction in relation to levels of data imbalance. In *Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA)*, vol. 1053 of *CEUR Workshop Proceedings*.
URL `http://ceur-ws.org/Vol-1053`

Gunn, S. (1998). Support vector machines for classification and regression. Project report.

Gupta, Y., Khan, Y., Gallaba, K., & McIntosh, S. (2017). The impact of the adoption of Continuous Integration on developer attraction and retention. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 491–494).

Hann, I.-H., Roberts, J., Slaughter, S., & Fielding, R. (2002). Economic incentives for participating in open source software projects. *International Conference on Information Systems*, (33).

Haruna, U., Maitalata, U. S., Mohammed, M., & Maitama, J. Z. (2021). Hausa intelligence chatbot system.

Hauff, C., & Gousios, G. (2015). Matching GitHub developer profiles to job advertisements. In *Working Conference on Mining Software Repositories*, (pp. 362–366). IEEE/ACM.

Haugeland, I. K. F., Følstad, A., Taylor, C., & Bjørkli, C. A. (2022). Understanding the user experience of customer service chatbots: An experimental study of chatbot interaction design. *International Journal of Human-Computer Studies*, *161*, 102788.

URL `https://www.sciencedirect.com/science/article/pii/S1071581922000179`

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284.

Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering (FOSE '07)*, (pp. 188–198).

High, R. (2012). The era of cognitive systems: An inside look at IBM Watson and how it works. *IBM Corporation, Redbooks*, *1*, 16.

Hildebrand, C., Hoffman, D. L., & Novak, T. P. (2021). Dehumanizing voice technology: Phonetic & experiential consequences of restricted human-machine interaction. *arXiv: Artificial Intelligence*.

Hillebrand, K., & Johannsen, F. (2021). KlimaKarl–a chatbot to promote employees' climate-friendly behavior in an office setting. In *International Conference on Design Science Research in Information Systems and Technology*, (pp. 3–15). Springer.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016a). Usage, costs, and benefits of continuous integration in open source projects. In *International Conference on Automated Software Engineering (ASE)*, (pp. 426–437). IEEE.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016b). Usage, costs, and benefits of continuous integration in open source projects. In *International Conference on Automated Software Engineering (ASE)*, (pp. 426–437). IEEE.

Hofmann, H., Kafadar, K., & Wickham, H. (2011). Letter-value plots: Boxplots for large data. Tech. rep., had.co.nz.

Horn, M., Li, X., Chen, L., Chen, L., Chen, L., & Kafle, S. (2021). A multi-talent healthcare AI bot platform.

Hu, Z., & Gehringer, E. F. (2019). Improving feedback on GitHub pull requests: A bots approach. In *IEEE Frontiers in Education Conference (FIE)*, (pp. 1–9).

Hurtado, S., Ray, P., & Marculescu, R. (2019). Bot detection in Reddit po-

litical discussion. In *Fourth International Workshop on Social Sensing*, SocialSense'19, (p. 30–35). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3313294.3313386`

Ilić, A., Ličina, A., & Savić, D. (2020). Chatbot development using Java tools and libraries. In *24th International Conference on Information Technology (IT)*, (pp. 1–4).

Ince, D., & Andrews, D. (1998). *The Software Life Cycle*. Butterworth.

Islam, M. R., & Zibran, M. F. (2017). Insights into Continuous Integration build failures. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 467–470).

Jaccard, P. (1912). The distribution of the flora in the alpine zone. *New Phytologist*, *11*(2), 37–50.

Jain, R., Singh, S., & Mishra, B. (2019). A brief study on build failures in Continuous Integration: Causation and effect. *Advances in Intelligent Systems and Computing*.

Jiménez, M., Piattini, M., & Vizcaíno, A. (2009). Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering*, *2009*.

Joblin, M., Eckl-Ganser, B., Bock, T., Schmid, A., Siegmund, J., & Apel, S. (2022). Hierarchical and hybrid organizational structures in open source software projects: A longitudinal study. *ACM Trans. Softw. Eng. Methodol.*. Just Accepted.
URL `https://doi.org/10.1145/3569949`

Josephs, A., Gilson, F., & Galster, M. (2022). Towards automatic classification of design decisions from developer conversations. In *IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, (pp. 10–14).

Just, S., Herzig, K., Czerwonka, J., & Murphy, B. (2016). Switching to git: The good, the bad, and the ugly. *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*.

Kalliamvakou, E., Damian, D., Singer, L., & German, D. M. (2014a). The codecentric collaboration perspective: Evidence from GitHub. Tech. rep.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014b). The promises and perils of mining GitHub. In *International Conference on Mining Software Repositories (MSR)*, (pp. 92–101). ACM.

Kalliamvakou, E., Gousios, G., Spinellis, D., & Nancy, P. (2009). Measuring developer contribution from software repository data. In *Mediterranean Conference on Information Systems*.

Kang, S., Liu, X. J., Kim, Y., & Yoon, V. (2021). Can bots help create knowledge? The effects of bot intervention in open collaboration. *Decision Support Systems*, *148*, 113601.
URL `https://www.sciencedirect.com/science/article/pii/S0167923621001111`

Kato, I. (1973). Development of WABOT 1. *Biomechanism*, *2*, 173–214.

Katz, J. (2020). Libraries.io open source repository and dependency metadata (version 1.6.0).
URL `https://doi.org/10.5281/zenodo.3626071`

Kaynak, I. K., Cilden, E., Çilden, E., & Aydin, S. (2019). Software quality improvement practices in Continuous Integration. *EuroSPI*.

Khanan, C., Luewichana, W., Pruktharathikoon, K., Jiarpakdee, J., Tantithamthavorn, C., Choetkiertikul, M., Ragkhitwetsagul, C., & Sunetnanta, T. (2020). JITBot: An explainable just-in-time defect prediction bot. In *35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, (p. 1336–1339). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3324884.3415295`

Kinsman, T., Wessel, M., Gerosa, M. A., & Treude, C. (2021). How do software developers use GitHub actions to automate their workflows? In *International Conference on Mining Software Repositories (MSR)*.

Koc, A., & Tansel, A. U. (2011). A survey of version control systems.

Kumar, R., Bansal, C., Maddila, C., Sharma, N., Martelock, S., & Bhargava, R. (2019). Building sankie: An AI platform for DevOps. In *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, (pp. 48–53).

Kuttal, S. K., Myers, J., Gurka, S., Magar, D., Piorkowski, D., & Bellamy, R. (2020). Towards designing conversational agents for pair programming: Accounting for creativity strategies and conversational styles. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, (pp. 1–11).

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, *33*(1), 159–174.
URL `http://www.jstor.org/stable/2529310`

Lanubile, F., Ebert, C., Prikladnicki, R., & Vizcaíno, A. (2010). Collaboration tools for global software engineering. *IEEE Software*, *27*(2), 52–55.

Laukkanen, E., & Mäntylä, M. (2015). Build waiting time in Continuous Integration – an initial interdisciplinary literature review. In *IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*, (pp. 1–4).

Laukkanen, E., Paasivaara, M., Itkonen, J., & Lassenius, C. (2018). Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering*.

Laurent, A. (2004). *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. O'Reilly Media.

Lebeuf, C., Storey, M. A., & Zagalsky, A. (2017). Software Bots. *IEEE Software*, *35*(1), 18–23.

Lebeuf, C., Zagalsky, A., Foucault, M., & Storey, M.-A. (2019). Defining and classifying software bots: A faceted taxonomy. In *International Workshop on Bots in Software Engineering*, (pp. 1–6).

Lebeuf, C. R. (2018). *A taxonomy of software bots: towards a deeper understanding of software bot characteristics*. Ph.D. thesis.

Lee, M., Frank, L. E., & IJsselsteijn, W. W. (2021). Brokerbot: A cryptocurrency chatbot in the social-technical gap of trust.

Leonard, A. (1998). *Bots: The Origin of New Species*. Penguin Books Limited.

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.-P., Itkonen, J., Mäntylä, M. V., & Männistö, T. (2015). The highways and country roads to continuous deployment. *IEEE Software*, *32*(2), 64–72.

Levenshtein, V. (1966). Binary codes capable of correcting deletions insertions and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Levy, K. (2014). Microsoft has its own version of Siri, a voice assistant called 'Cortana'. `https://www.businessinsider.com/microsofts-cortana-voice-assistant-2014-4`. Accessed: 2022-07-14.

Li, C., Li, C., Xing, W., Xing, W., & Xing, W. (2021). Natural language generation using deep learning to support MOOC learners. *International Journal of Artificial Intelligence in Education*.

Liao, Z., Qi, X., Zhang, Y., Fan, X., & Zhou, Y. (2020). How to evaluate the productivity of software ecosystem: A case study in GitHub. *Scientific Programming*.

Lim, W. M., Kumar, S., Verma, S., & Chaturvedi, R. (2022). Alexa, what do we know about conversational commerce? Insights from a systematic literature review. *Psychology & Marketing*.

Lin, X., Shao, B., & Wang, X. (2022). Employees perceptions of chatbots in B2B marketing: Affordances vs. disaffordances. *Industrial Marketing Management*, *101*, 45–56.
URL `https://www.sciencedirect.com/science/article/pii/S001985012100242X`

Lindley, D. V. (1990). *Thomas Bayes*, (pp. 10–11). London, UK: Palgrave Macmillan.

Liu, D., Smith, M. J., & Veeramachaneni, K. (2020). Understanding user-bot interactions for small-scale automation in open-source development. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, (p. 1–8). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3334480.3382998`

Long, K., Vines, J., Sutton, S., Brooker, P., Feltwell, T., Kirman, B., Barnett, J., & Lawson, S. (2017). "could you define that in bot terms"? Requesting, creating and using bots on Reddit. In *CHI Conference on Human Factors in Computing Systems*, CHI '17, (p. 3488–3500). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3025453.3025830`

Lunden, I. (2014). Amazon gets into voice recognition, buys Ivona software to compete against Apple's Siri. `https://techcrunch.com/2013/01/24/amazon-gets-into-voice-recognition-buys-ivona-software-to-.compete-against-apples-siri/`. Accessed: 2022-07-14.

Ma, Y., Bogart, C., Amreen, S., Zaretzki, R., & Mockus, A. (2019). World of code: An infrastructure for mining the universe of open source VCS data. In *International Conference on Mining Software Repositories*, (pp. 143–154). IEEE Press.
URL `https://doi.org/10.1109/MSR.2019.00031`

Manglaviti, M., Coronado-Montoya, E., Gallaba, K., & McIntosh, S. (2017). An empirical study of the personnel overhead of Continuous Integration. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 471–474).

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, (pp. 50–60).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Martensson, T., Ståhl, D., & Bosch, J. (2019). Test activities in the continuous integration and delivery pipeline.

Matthies, C., Dobrigkeit, F., & Hesse, G. (2019). An additional set of (automated) eyes: Chatbots for agile retrospectives.

Mauldin, M. L. (1994). Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI*, vol. 94, (pp. 16–21).

Maus, G. (2017). A typology of socialbots (abbrev.). In *ACM on Web Science Conference*, WebSci '17, (p. 399–400). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3091478.3098860`

Mccallum, A., & Nigam, K. (2001). A Comparison of Event Models for Naive Bayes Text Classification. *Work Learn Text Categ*, *752*.

McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia medica*, *22*(3), 276–282.
URL `https://pubmed.ncbi.nlm.nih.gov/23092060https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/`

Mendes Samagaio, Á., Lopes Cardoso, H., & Ribeiro, D. (2021). A chatbot for recipe recommendation and preference modeling. In G. Marreiros, F. S. Melo, N. Lau, H. Lopes Cardoso, & L. P. Reis (Eds.) *Progress in Artificial Intelligence*, (pp. 389–402). Cham: Springer International Publishing.

Mendoza, M., Tesconi, M., & Cresci, S. (2020). Bots in social and interaction networks: Detection and impact estimation. *ACM Trans. Inf. Syst.*, *39*(1). URL `https://doi.org/10.1145/3419369`

Metz, C. (2015). How GitHub conquered Google, Microsoft, and everyone else.
URL `https://www.wired.com/2015/03/github-conquered-google-/microsoft-everyone-else/`

Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013). Efficient estimation of word representations in vector space.
URL `http://arxiv.org/abs/1301.3781`

Minnich, A., Chavoshi, N., Koutra, D., & Mueen, A. (2017). BotWalk: Efficient adaptive exploration of twitter bot networks. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, (pp. 467–474).

Minutolo, A., Damiano, E., Pietro, G. D., Fujita, H., & Esposito, M. (2021). A conversational agent for querying italian patient information leaflets and improving health literacy. *Computers in Biology and Medicine*.

Mirhosseini, S., & Parnin, C. (2017). Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *International Conference on Automated Software Engineering ({ASE})*, (pp. 84–94).

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Moharil, A., Orlov, D., Jameel, S., Trouwen, T., Cassee, N., & Serebrenik, A. (2022). Between JIRA and GitHub: ASFBot and its influence on human comments in issue trackers. In *Mining Software Repositories*.

Mohayeji Nasrabadi, H., Ebert, F., Arts, E., Constantinou, E., & Serebrenik, A. (2022). On the adoption of a TODO bot on GitHub: A preliminary study. International Workshop on Bots in Software Engineering, BotSE ; Conference date: 09-05-2022 Through 09-05-2022.
URL `http://botse.org/`

Molnár, G., & Szüts, Z. (2018). The role of chatbots in formal education. In *IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, (pp. 000197–000202).

Monperrus, M. (2019). Explainable software bot contributions: Case study of automated bug fixes. *IEEE/ACM 1st International Workshop on Bots in Software Engineering, BotSE 2019*, (pp. 12–15).

Monperrus, M., Urli, S., Durieux, T., Martinez, M., Baudry, B., & Seinturier, L. (2019). Repairnator patches programs automatically. *Ubiquity*, *2019*(July), 1–12.

Motger, Q., Franch, X., & Marco, J. (2021). Conversational agents in software engineering: Survey, taxonomy and challenges. *arXiv: Computation and Language*.

Mowforth, P., & Bratko, I. (1987). AI and robotics; flexibility and integration. *Robotica*, *5*(2), 93–98.

Mårtensson, T., Ståhl, D., & Bosch, J. (2017). Continuous Integration impediments in large-scale industry projects. In *IEEE International Conference on Software Architecture (ICSA)*, (pp. 169–178).

Nakagawa, T., Higo, Y., & Kusumoto, S. (2020). Clione: Clone modification support for pull request based development. In *27th Asia-Pacific Software Engineering Conference (APSEC)*, (pp. 455–459).

Nalini, C., Kumari, R. S., Parteban, G. K., Priyaa, T. N., & Sanchay, A. S. (2021). AI based chatbot in food industry.

Neely, S., & Stolt, S. (2013). Continuous Delivery? Easy! Just change everything (well, maybe it is not that easy). In *2013 Agile Conference*, (pp. 121–128).

Ni, Z., Shen, B., Chen, Y., Meng, Z., Meng, Z., Meng, Z., & Cao, J. (2019). CrowDevBot: A task-oriented conversational bot for software crowdsourcing platform(s).

Oguntosin, V., & Olomo, A. (2021). Development of an E-Commerce chatbot for a university shopping mall. *Applied Computational Intelligence and Soft Computing*.

Okanović, D., Beck, S., Merz, L., Zorn, C., Merino, L., van Hoorn, A., & Beck,

F. (2020). Can a chatbot support software engineers with load testing? Approach and experiences. In *ACM/SPEC International Conference on Performance Engineering*, ICPE '20, (p. 120–129). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3358960.3375792`

Ortu, M., Pinna, A., Tonelli, R., Marchesi, M., Bowes, D., & Destefanis, G. (2018). Angry-Builds: An empirical study of affect metrics and builds success on GitHub ecosystem. In *19th International Conference on Agile Software Development: Companion*, XP '18. New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3234152.3234160`

Otte, S. (2009). Version Control Systems.

Paikari, E., Choi, J., Kim, S., Baek, S., Kim, M., Lee, S., Han, C., Kim, Y., Ahn, K., Cheong, C., & Van Der Hoek, A. (2019). A chatbot for conflict detection and resolution. *IEEE/ACM 1st International Workshop on Bots in Software Engineering, BotSE 2019*, (pp. 29–33).

Paikari, E., & Van Der Hoek, A. (2018). A framework for understanding chatbots and their future. *International Conference on Software Engineering*, (pp. 13–16).

Paixão, K. V. R., Felício, C. Z., Delfim, F. M., & De A. Maia, M. (2017). On the interplay between non-functional requirements and builds on Continuous Integration. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 479–482).

Park, D., Cho, H., & Lee, S. (2022). Classifying issues into custom labels in GitBot. In *4th International Workshop on Bots in Software Engineering, ICSE'22*.

Patsoulis, G., Promikyridis, R., & Tambouris, E. (2021). Integration of chatbots with knowledge graphs in E-Government: The case of getting a passport. In *25th Pan-Hellenic Conference on Informatics*, PCI 2021, (p. 425–429). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3503823.3503901`

Paule, C., Düllmann, T. F., & Van Hoorn, A. (2019). Vulnerabilities in Continuous Delivery pipelines? A case study. In *IEEE International Conference on Software Architecture Companion (ICSA-C)*, (pp. 102–108).

Pawlik, V. P. (2022). Design matters! how visual gendered anthropomorphic design cues moderate the determinants of the behavioral intention towards using chatbots. In *Chatbot Research and Design*, (pp. 192–208). Cham: Springer International Publishing.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peérez-Soler, S., Gonzaélez-Jimeénez, M., Guerra, E., & de Lara, J. (2019). Towards conversational syntax for domain-specific languages using chatbots. *Journal of Object Technology*, *18*(2), 5:1–21. The 15th European Conference on Modelling Foundations and Applications.
URL `http://www.jot.fm/contents/issue_2019_02/article5.html`

Pham, R., Singer, L., Liskin, O., Filho, F. F., & Schneider, K. (2013). Creating a shared understanding of testing culture on a social coding site. In *35th International Conference on Software Engineering (ICSE)*, (pp. 112–121).

Pinheiro, A. M., Rabello, C. S., Furtado, L. B., Pinto, G., & de Souza, C. R. (2019). Expecting the unexpected: Distilling bot development, challenges, and motivations. In *IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, (pp. 51–52).

Pinto, G., Castor, F., Bonifácio, R., & Reboucas, M. (2018). Work practices and challenges in continuous integration: A survey with Travis CI users. *Software - Practice and Experience*.

Qasse, I., Mishra, S., & Hamdaqa, M. (2021). iContractBot: A chatbot for smart contracts' specification and code generation. In *IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, (pp. 35–38).

Rahman, A., Agrawal, A., Krishna, R., & Sobran, A. (2018). Characterizing the influence of Continuous Integration: Empirical results from 250+ open source and proprietary projects. In *4th ACM SIGSOFT International Workshop on Software Analytics*, SWAN 2018, (p. 8–14). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3278142.3278149`

Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015). Synthesizing Continuous Deployment practices used in software development. In *2015 Agile Conference*, (pp. 1–10).

Rahman, M. M., & Roy, C. K. (2017). Impact of Continuous Integration on Code Reviews. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 499–502).

Rajaobelina, L., & Ricard, L. (2021). Classifying potential users of live chat services and chatbots. *Journal of Financial Services Marketing*.

Ramesh, K., Ravishankaran, S., Joshi, A., Chandrasekaran, K., & Chandrasekaran, K. (2017). A survey of design techniques for conversational agents.

Ranoliya, B. R., Raghuwanshi, N., & Singh, S. (2017). Chatbot for university related FAQs. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, (pp. 1525–1530).

Rausch, T., Hummer, W., Leitner, P., & Schulte, S. (2017). An empirical analysis of build failures in the Continuous Integration workflows of Java-based open source software. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 345–355).

Rebai, S., Ben Sghaier, O., Alizadeh, V., Kessentini, M., & Chater, M. (2019). Interactive refactoring documentation bot. In *International Working Conference on Source Code Analysis and Manipulation (SCAM)*, (pp. 152–162). IEEE.

Reier Forradellas, R. F., & Garay Gallastegui, L. M. (2021). Digital transformation and artificial intelligence applied to business: Legal regulations, economic impact and perspective. *Laws*, *10*(3).
URL `https://www.mdpi.com/2075-471X/10/3/70`

Ren, R., Castro, J. W., Santos, A., Pérez-Soler, S., Acuña, S. T., & de Lara, J. (2020). Collaborative modelling: Chatbots or on-line tools? An experimental study. In *Evaluation and Assessment in Software Engineering*, EASE '20, (p. 260–269). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3383219.3383246`

Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE Software*, *26*(2), 52–58.

Rochkind, M. J. (1975). The source code control system. *IEEE Trans. Softw. Eng.*, *1*(1), 364–370.
URL `https://doi.org/10.1109/TSE.1975.6312866`

Rodríguez, P., Mikkonen, K., Kuvaja, P., Oivo, M., & Garbajosa, J. (2013). Building lean thinking in a telecom software development organization: Strengths and challenges. In *International Conference on Software and System Process*, ICSSP 2013, (p. 98–107). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2486046.2486064`

Rodríguez-Ruiz, J., Mata-Sánchez, J. I., Monroy, R., Loyola-González, O., & López-Cuevas, A. (2020). A one-class classification approach for bot detection on Twitter. *Computers & Security*, *91*, 101715.

Romano, J., Kromrey, J. D., Coraggio, J., Skowronek, J., & Devine, L. (2006). Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices? In *Annual Meeting of the Southern Association for Institutional Research*.

Rombaut, B., Cogo, F. R., Adams, B., & Hassan, A. E. (2023). There's no such thing as a free lunch: Lessons learned from exploring the overhead introduced by the Greenkeeper dependency bot in npm. *ACM Trans. Softw. Eng. Methodol.*, *32*(1).
URL `https://doi.org/10.1145/3522587`

Rossi, C., Shibley, E., Su, S., Beck, K., Savor, T., & Stumm, M. (2016). Continuous Deployment of mobile software at Facebook (showcase). In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, (p. 12–23). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2950290.2994157`

Saadat, S., Colmenares, N., & Sukthankar, G. (2021). Do bots modify the workflow of GitHub teams?

Safavian, S., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, *21*, 660–674.

Sagstad, M. H., Morken, N.-H., Lund, A., Dingsør, L. J., Nilsen, A. B. V., & Sørbye, L. M. (2022). Quantitative user data from a chatbot developed for women with gestational diabetes mellitus: Observational study. *JMIR formative research*.

Saini, R., Mussbacher, G., Guo, J. L. C., & Kienzle, J. (2020). *DoMoBOT: A Bot for Automated and Interactive Domain Modelling*. New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3417990.3421385`

Sampedro, Z., Holt, A., & Hauser, T. (2018). Continuous Integration and Delivery for HPC: Using Singularity and Jenkins. In *Practice and Experience on Advanced Research Computing*, PEARC '18. New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3219104.3219147`

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., & Stumm, M. (2016). Continuous deployment at Facebook and OANDA. In *International Conference on Software Engineering (ICSE)*, (pp. 21–30). IEEE.

Sbaitso, D. (1991). Dr. sbaitso. `https://classicreload.com/dr-sbaitso.html`.

Schanke, S., Burtch, G., & Ray, G. (2021). Estimating the impact of "humanizing" customer service chatbots. *Information Systems Research*.

Schanke, S. D. (2021). *Humanizing Digital Experiences: Three Essays on the Design of Digital Entities*. Ph.D. thesis, University of Minnesota.

Schueller, W., & Wachs, J. (2022). Modeling interconnected social and technical risks in open source software ecosystems.
URL `https://arxiv.org/abs/2205.04268`

Schueller, W., Wachs, J., Servedio, V., Thurner, S., & Loreto, V. (2022). Evolving collaboration, dependencies, and use in the rust open source software ecosystem. *Scientific Data*.

Seiler, R., & Schär, A. (2021). Chatbots, conversational interfaces, and the stereotype content model.

Serban, D., Golsteijn, B., Holdorp, R., & Serebrenik, A. (2021). SAW-BOT: Proposing fixes for static analysis warnings with GitHub suggestions. In *IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, (pp. 26–30).

Serrano Alves, L. P., Wiese, I. S., Chaves, A. P., & Steinmacher, I. (2022). How to find my task? Chatbot to assist newcomers in choosing tasks in OSS projects. In *Chatbot Research and Design*, (pp. 90–107). Cham: Springer International Publishing.

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, *5*, 3909–3943.

Shawar, B. A., & Atwell, E. (2015). Alice chatbot: Trials and outputs. *Computación y Sistemas*, *19*, 625 – 632.
URL `http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-55462015000400625&nrm=iso`

Siedlikowski, S., Noël, L.-P., Moynihan, S. A., & Robin, M. (2021). Chloe for COVID-19: Evolution of an intelligent conversational agent to address infodemic management needs during the COVID-19 pandemic. *Journal of Medical Internet Research*.

Skelton, M., & O'Dell, C. (2016). *Continuous Delivery with Windows and .NET*. Oreilly.
URL `https://www.oreilly.com/library/view/continuous-delivery-with/9781492042327/`

Somasundaram, R. (2013). *Git: Version control for everyone*. Packt Publishing.

Souza, R., & Silva, B. (2017). Sentiment analysis of Travis CI builds. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 459–462).

Sparck Jones, K. (1972). *A Statistical Interpretation of Term Specificity and Its Application in Retrieval*, (pp. 11–21). MCB UP Ltd.

Ståhl, D., & Bosch, J. (2014). Automated software integration flows in industry: A multiple-case study. In *36th International Conference on Software Engineering*, (pp. 54–63).

Statt, N. (2016). Why Google's fancy new AI assistant is just called 'Google'. `https://www.theverge.com/2016/5/20/11721278/google-ai-assistant-name-vs-alexa-siri`. Accessed: 2022-06-20.

Steinmacher, I., Treude, C., & Gerosa, M. A. (2019). Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software*, *36*(4), 41–49.

Storey, M. A., & Zagalsky, A. (2016). Disrupting developer productivity one Bot at a Time. *ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, (pp. 928–931).

Ståhl, D., & Bosch, J. (2016). Industry application of Continuous Integration modeling: A multiple-case study. In *38th International Conference on Software Engineering Companion*, ICSE '16, (p. 270–279). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2889160.2889252`

Ståhl, D., & Bosch, J. (2013). Experienced benefits of continuous integration in industry software product development: A case study. *ICSE 2013*.

Ståhl, D., Mårtensson, T., & Bosch, J. (2017). The continuity of continuous integration: Correlations and consequences. *Journal of Systems and Software*, *127*, 150–167.
URL `https://www.sciencedirect.com/science/article/pii/S0164121217300328`

Subramanian, V., Ramachandra, N., & Dubash, N. (2019). TutorBot: Contextual learning guide for software engineers. In *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, (pp. 16–17).

Sultanía, A. K. (2015). Developing software product and test automation software using agile methodology. In *Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, (pp. 1–4).

Thistlethwaite, D. L., & Campbell, D. T. (1960). Regression-discontinuity analysis: An alternative to the ex post facto experiment. *Journal of Educational Psychology*, *51*(6), 309–317.

Thomas, N. T. (2016). An e-business chatbot using AIML and LSA.

Tian, Y., Thung, F., Sharma, A., & Lo, D. (2017). APIBot: Question answering bot for API documentation. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (pp. 153–158).

Tichy, W. F. (1985). Rcs — a system for version control. *Software: Practice and Experience*, *15*(7), 637–654.
URL `https://doi.org/10.1002/spe.4380150703`

Tsay, J., Dabbish, L., & Herbsleb, J. (2014a). Influence of social and technical factors for evaluating contribution in GitHub. In *International Conference on Software Engineering (ICSE)*, (pp. 356–366). ACM.

Tsay, J., Dabbish, L., & Herbsleb, J. (2014b). Let's talk about it: Evaluating contributions through discussion in GitHub. In *ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE)*, (pp. 144–154). ACM.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *LIX*(236), 433–460.

Tymchuk, Y., Mocci, A., & Lanza, M. (2014). Collaboration in open source projects: Myth or reality? In *11th Working Conference on Mining Software Repositories*, MSR 2014, (p. 304–307). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2597073.2597093`

Urli, S., Yu, Z., Seinturier, L., & Monperrus, M. (2018). How to design a program repair bot?: Insights from the repairnator project. *International Conference on Software Engineering (ICSE)*, (pp. 95–104).

Valenzuela-Toledo, P., & Bergel, A. (2022). Evolution of GitHub action workflows. In *29th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE.

Varuna, T. V., & Mohan, A. (2019). Trend prediction of GitHub using time series analysis. In *10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, (pp. 1–7).

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G., Serebrenik, A., Devanbu, P., & Filkov, V. (2015a). Gender and tenure diversity in GitHub teams. In *Human Factors in Computing Systems (CHI)*, (pp. 3789–3798). ACM.

Vasilescu, B., van Schuylenburg, S., Wulms, J., Serebrenik, A., & Brand, M. (2014). Continuous Integration in a social-coding world: Empirical evidence from GitHub.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., & Filkov, V. (2015b). Quality and productivity outcomes relating to Continuous Integration in GitHub. In *10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (p. 805–816). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/2786805.2786850`

Vassallo, C., Palomba, F., Bacchelli, A., & Gall, H. C. (2018). *Continuous Code Quality: Are We (Really) Doing That?*, (p. 790–795). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3238147.3240729`

Vassallo, C., Proksch, S., Gall, H. C., & Di Penta, M. (2019). Automated reporting of anti-patterns and decay in Continuous Integration. In *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, (pp. 105–115). IEEE.

Vassallo, C., Schermann, G., Zampetti, F., Romano, D., Leitner, P., Zaidman, A., Di Penta, M., & Panichella, S. (2017). A tale of CI build failures: An open source and a financial organization perspective. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 183–193).

Vassallo, C., Zampetti, F., Romano, D., Beller, M., Panichella, A., Di Penta, M., & Zaidman, A. (2016). Continuous Delivery practices in a large financial organization. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 519–528).

Wallace, R. (1995). Artificial linguistic internet computer entity (alice).

Wang, Z., Wang, Y., & Redmiles, D. (2022). From specialized mechanics to project butlers: The usage of bots in open source software development. *IEEE Software*, *39*(5), 38–43.

Weber, I., Nepal, S., & Zhu, L. (2016). Developing dependable and secure cloud applications. *IEEE Internet Computing*, *20*(3), 74–79.

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, *9*(1), 36–45.
URL `https://doi.org/10.1145/365153.365168`

Wessel, M. (2020). *Enhancing Developers Support on Pull Requests Activities with Software Bots*, (p. 1674–1677). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3368089.3418539`

Wessel, M., de Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018a). The power of bots: Characterizing and understanding bots in OSS projects. *Human-Computer Interaction*, *2*.

Wessel, M., De Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018b). The power of bots: Understanding bots in OSS projects. *The ACM International Conference on Human-Computer Interaction*.

Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2020). What to expect from code review bots on GitHub? A survey with OSS maintainers. In *34th Brazilian Symposium on Software Engineering*, SBES '20, (p. 457–462). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3422392.3422459`

Wessel, M., & Steinmacher, I. (2020). The inconvenient side of software bots on pull requests. In *International Workshop on Bots in Software Engineering (BotSE)*.

Wessel, M., Steinmacher, I., Wiese, I., & Gerosa, M. A. (2019). Should i stale or should i close?: An analysis of a bot that closes abandoned issues and pull requests. In *1st International Workshop on Bots in Software Engineering*, (pp. 38–42). IEEE Press.

Wessel, M. S. (2021). *Perception of software bots on pull requests on social coding environments*. Ph.D. thesis, Universidade dee Sao Paulo-Instituto de Matemática e Estatística.

Whitehead, J. (2007). Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE '07)*, (pp. 214–225).

Widder, D., Vasilescu, B., Hilton, M., & Kästner, C. (2018). I'm leaving you, Travis: A continuous integration breakup story. In *IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, (pp. 165–169). IEEE.

Widder, D. G., Hilton, M., Kästner, C., & Vasilescu, B. (2019). A conceptual replication of Continuous Integration pain points in the context of Travis CI. In *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, (p. 647–658). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3338906.3338922`

Winarsky, N., Mark, B., & Kressel, H. (2012). The development of Siri and the SRI venture creation process. *SRI International*.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 3rd ed.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., & Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Wood, A., Eberhart, Z., & McMillan, C. (2020). *Dialogue Act Classification for Virtual Agents for Software Engineers during Debugging*, (p. 462–469). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3387940.3391487`

Wu, X., Gao, A., Zhang, Y., Wang, T., & Tang, Y. (2022). A preliminary study of bots usage in open source community. In *13th Asia-Pacific Symposium on Internetware*, Internetware '22, (p. 175–180). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3545258.3545284`

Wyrich, M., & Bogner, J. (2019). Towards an autonomous bot for automatic source code refactoring. *1st International Workshop on Bots in Software Engineering*, (pp. 24–28).

Wyrich, M., Ghit, R., Haller, T., & Müller, C. (2021). Bots don't mind waiting, do they? Comparing the interaction with automatically and manually created pull requests. In *IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, (pp. 6–10).

Xu, A., Liu, Z., Guo, Y., Sinha, V., & Akkiraju, R. (2017a). A new chatbot for customer service on social media. In *CHI Conference on Human Factors in Computing Systems*, CHI '17, (p. 3506–3510). New York, NY, USA: Association for Computing Machinery.
URL `https://doi.org/10.1145/3025453.3025496`

Xu, B., Xing, Z., Xia, X., & Lo, D. (2017b). AnswerBot: Automated generation of answer summary to developers' technical questions. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (pp. 706–716).

Young, J.-G., Casari, A., McLaughlin, K., Trujillo, M. Z., Hébert-Dufresne, L., & P. Bagrow, J. (2021). Which contributions count? Analysis of attribution in open source. In *International Conference on Mining Software Repositories*, (pp. 242–253). IEEE/ACM.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., & Vasilescu, B. (2015). Wait for it: Determinants of pull request evaluation latency on GitHub. In *IEEE/ACM 12th Working Conference on Mining Software Repositories*, (pp. 367–371).

Zampetti, F., Bavota, G., Canfora, G., & Penta, M. D. (2019). A study on the interplay between pull request review and Continuous Integration builds. In *IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (pp. 38–48).

Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., & Di Penta, M. (2017). How open source projects use static code analysis tools in continuous integration pipelines. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, (pp. 334–344). IEEE.

Zhai, H., Casalnuovo, C., & Devanbu, P. (2019). Test coverage in Python programs. In *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, (pp. 116–120).

Zhang, N., Huang, Q., Xia, X., Zou, Y., Lo, D., & Xing, Z. (2022a). Chatbot4QR: Interactive query refinement for technical question retrieval. *IEEE Transactions on Software Engineering*, *48*(4), 1185–1211.

Zhang, X., Yu, Y., Wang, T., Rastogi, A., & Wang, H. (2022b). Pull request latency explained: An empirical overview. *Empirical Software Engineering*, *27*(6), 1–38.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., & Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (pp. 60–71).

Zheng, L. N., Albano, C. M., Vora, N. M., Mai, F., & Nickerson, J. V. (2019). The roles bots play in Wikipedia. *Proc. ACM Hum.-Comput. Interact.*, *3*(CSCW).
URL `https://doi.org/10.1145/3359317`

Čapek, K. (1920). *R.U.R. (Rossum's universal robots)*.
URL `https://livablesoftware.com/best-bots-software-development/`