Chair of Software Engineering

Understanding How Developers Link Social and Technical Assets in GitHub

Masterarbeit von

Arun Kumar Verma

1. Prüfer	2. Prüfer
Dr.-Ing. Sven Apel	Prof. Dr. Gordon Fraser

15. März 2020

Contents

Abstract

Collaborative software development is a team effort in which success depends on the ability to coordinate social and technical assets. Version control systems and social network platforms were created to support technical and social assets management. While version control systems keep track of code changes (i.e., technical assets), social network platforms present a common interface to not only host the project source code, but also to provide a communication channel among developers (i.e., social assets). On GitHub social coding platform, developers may link the problem they are solving (i.e., the issue) with the solution (i.e., the pull requests). Considering that the pull-based software development model has well-defined steps and GitHub is able to support these steps, GitHub becomes an interesting social coding platform to be investigated. Our goal is to investigate how developers use GitHub facilities for social and technical assets coordination in practice. To achieve this goal, we performed a two-steps study. First, we empirically investigate how developers organize and link social and technical assets investigating 66 GitHub repositories. Second, we ask 100 developers about discrepancies in the theoretical and practice of the pull-based model, and challenges they have when contributing to this model. Our results show: (i) developers link 24.67% of the problems (issues) with related problems and with proposed solutions (pull requests), (ii) developers link commits with both problems and solutions in 84.26% of the cases, and (iii) developers link 59.43% of issues with labels, and 61.60% of pull requests with labels. The results of our survey for the issues and pull requests relationship show that: (i) 59% of the participants link issues with related issues, (ii) 58% of the participants refer issues in the body of pull requests, and (iii) 57% of the participants describe pull requests in the description of issues. Regarding the relationship among issues and commits, we found that: (i) 84% of the participants refer commits in the description of issues, and (ii) 94% of the participants link issues with commit messages. Regarding the relationship among issues and labels, we found that 63% of the participants link issues and pull requests with labels. In addition, we found that 80% of the participants agreed that relating assets are useful for project coordination, but they do not create links very often in practice. The take away message is that: (i) most of the links do not come from the natural flow (from problem description to solution), the links come from the opposite flow (i.e., from the solution to the problem description). Hence, developers should pay more attention to link issues and pull requests in both directions to increase coordination among developers; (ii) linking labels with issues and pull requests may help developers to refine communication related to code changes. Hence, we encourage developers to link issues and pull requests with labels; and, (iii) despite developers agree that they need to link social and technical assets, they do not link assets frequently in practice. This way, we recommend that they create links more often.

Acknowledgments

This thesis becomes a reality with the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

I express my immense gratitude towards my family and friends for their kind support and encouragement which helped me towards the completion of this master thesis.

I am highly indebted to the **University of Passau** and **Faculty of Informatics and Mathematics** for their guidance and constant supervision as well as for providing necessary information regarding this research & also for their support in completing this endeavor.

I would like to express my special gratitude and thanks to my adviser, **Gustavo do Vale** for his guidance and support throughout the thesis work. Without his encouragement guidance, this thesis would not have materialized.

My thanks and appreciations also go to my colleague and people who have willingly helped me out with their abilities.

List of Figures

List of Tables

1 Introduction

Software development is a collaborative and distributed activity in which success depends on the ability to coordinate social and technical assets [JAM17]. Poor coordination among developers can reduce the quality of a product because developers need information from each other in order to carry out their tasks well. In software development, effective coordination is a fundamental requirement because it helps team members to work together, and gives them the opportunity to share their point of view [Hel18]. There are several coordination channels available for developers such as emails, or writing a shared document. Social media play an important role in supporting coordination activities within a community of software developers. According to Souza et al. [Sou+04] the success of software projects depends on coordination among contributors. Bird et al. [Bir+09] and Sedano et at. [SRP17] also mentioned that the lack of coordination is a critical problem in distributed software development. In summary, if coordination is uncertain or inaccurate, it may compromise the project budget and schedule.

1.1 Motivation

Although software developers make code changes and craft software, they engage in various types of conversations such as discussion with other developers about bug fixing or giving feedback for a code review. The pull-based development is an emerging paradigm for distributed software development that supports developers with isolated development and branching techniques. Therefore, many projects are being migrated to various code hosting platforms such as GitHub and Bitbucket [GPD14]. Git stores source code and keeps track of the code changes history. GitHub provides a web interface to Git repositories, facilitates social coding, and supports social networking among developers. Of course, there is common a large amount of communication in the development of a project, however, only when the communication is linked with code artifacts it will be useful for further analyses and to increase the developers' understanding. Considering it, tools that facilitate such integration and coordination among technical and social assets are well regarded in software development, specially in the pull-based software development [GPD14]. GitHub provides many features to link social and technical assets such as issues, pull requests, commits, labels. On GitHub, each issue may address one task and depending on the scope of the issue further issues are created. A pull request allows developers to start a discussion or a code review. Developers should map pull requests with relevant issues (usually referred in pull request description) [RR14]. Furthermore, developers may assign relevant labels to issues while creating them to facilitate GitHub grouping or filtering of issues and pull requests.

1.2 Goal

The main goals of this thesis are: (i) to understand which GitHub features developers use to support their coordination, and (ii) to know how and when developers normally use all GitHub features to integrate social and technical assets. The understanding of how developers link social and technical assets can be useful to guide developers of open-source projects to follow the best practices. Furthermore, it can also be useful for researchers to refine relationship among social and technical assets when analyzing the communication of open-source projects. In other words, they can include only communication that is related to the source code changes of the project.

1.3 Methodology

To achieve our goals we performed a two-step study. First, we performed an empirical study mining 66 GitHub repositories and collecting information about GitHub issues, pull requests, commits, and labels of each repository. With this information, we investigate three relations among assets in both directions: (i) whether developers normally link the problem description (i.e., a GitHub issue) with the proposed solution (i.e., a GitHub pull request), (ii) whether developers link commits with the problem and with their solution and, (iii) whether developers use labels to represent the type of issues they are working on such as bug, feature, question, and maintenance. Second, we surveyed 100 developers to find out how often developers link social and technical assets in practice. This two-step study is necessary because while the first presents the working practice, the second investigates developers' opinions about linking social and technical assets on GitHub. Therefore, only with these two studies, we are able to provide guidelines that will support practitioners coordinating their assets in GitHub.

1.4 Results

Our results for the first relationship (i.e., issues and pull requests) show that developers link 24.67% of the problems with related problems. Following the natural flow from problems to the solutions, we found that only 6.74% of the issues are linked with pull requests. In other direction, we retrieved that 20.55% of pull requests are linked with issues. For the second relationship (i.e., issues and commits), we see that 84.26% of the commits are linked with issues, but in the other direction we found that 50.58% of the issues are linked with commits. Regarding links among commits and pull requests, developers link 80.92% of the commits in pull requests. On the other direction, we found that 89.28% of the pull requests in the commit messages. For the third relationship (i.e., issues and labels), we found that 59.43% of the issues are linked with labels and that 61.60% of the pull request are linked with labels.

1.5 Discussion

We conducted a survey with 100 valid responses to investigate developers' opinion about linking social and technical assets. Regarding the relationship among issues and pull requests, we found that (i) 59% of the participants link issues with related issues, (ii) 58% of the participants refer issues in the body of pull requests, and (iii) 57% of the participants describe pull requests in the description of issues. Regarding the relationship among issues and commits, we found that 84% of the participants refer commits in the description of issues, and 94% of the participants link issues with commit messages. Regarding the relationship among issues and labels, we found that the participants use both default and customized label and assign them to issues and pull requests in 63% of the cases. Participants reported that relating commits with issues in both directions is useful for retrieving code changes. The results from the survey are similar to what we found empirically. Around 80% of the participants agreed that links among issues, pull requests, commits, and labels are useful, however, they do not relate social and technical assets very often in practice.

1.6 Contributions

This thesis makes the following main contributions:

- We empirically present evidence that one quarter (24.67%) of the developers link problem descriptions with solutions.
- We provide evidence that 84.26% of the developers refer commits in the description of issues.
- We found that developers link 59.43% of the issues with labels and 61.60% of the pull requests with labels.
- We show that 80% of the participants of our survey agreed that linking social and technical assets is important for project coordination, however, they do not link assets very often in practice.
- We provide a list of the top 10 labels used in general, in issues (not pull requests), and in pull requests. As result, we found that: *cla: yes, CLA Signed* labels are the two most related to code (pull requests). *FrozenDueToAge, question* labels are the two most related to communication (issues). And, *bug, needs_traige, module* labels are related to both issues and pull requests. Assigning labels to assets help developers to refine communication related to code changes.
- We make our infrastructure publicly available, and the data is also publicly available [Ver20].

1.7 Structure of the Thesis

The rest of this thesis is structured as follows. In chapter 2, we describe background on collaborative software development, pull-based software development model, version control systems, and communication and coordination of developers on collaborative software development. In chapter 3, we present related work that investigate coordination and communication on software development. In chapter 4, we present our methodology showing research questions, subject projects, how we get data, and how we operationalize our analysis. In chapter 5, we present the results obtained to answer research questions. In chapter 6, we present a comparison of our results with the result of previous studies, a survey to investigate developers' opinion about linking social and technical assets, and threats to the validity our study. Finally, in chapter 7, we conclude the thesis and present suggestions for future work.

2 Background

This chapter presents important concepts to understand this thesis. In Section 2.1, we present the benefits and challenges of collaborative software development. In Section 2.2, we present details about pull-based software development model. In Section 2.3, we provide basic information about version control systems. In Section 2.4, we present details about developers' communication on collaborative software development. In Section 2.5, we present the developers' coordination in software development, and how they organize their social and technical assets. In Section 2.6 and 2.7, we present the communication flow on GitHub and how developers may relate GitHub features.

2.1 Collaborative Software Development

There are lots of factors that influence the success of a project such as communication media, coordination tools. Having a team full of great individuals will not bring the desired results if there is not effective collaboration. When developing software, there is a huge flow of information. It is therefore important for team coordination to be effective and timely. Good coordination saves time, money, and effort [Inj18]. Poor co-ordination (or no collaboration) threatens software development project success [Nes19]. The challenges of developing an application or software as a team may be reduced by using groupware to coordinate and communicate the complex details concerned within a project. When developers are geographically separated, their coordination success may depend on utilizing effective groupware applications [Thi+07]. Groupware is an application designed to assist developers concerned in a common task to reach their goals [JJ91].

Depending on the level of collaboration, groupware can be divided into three categories as communication, conferencing, and coordination. Communication is an unstructured interchange of information such as a phone call or a chat. Conferencing can be thought of as interactive work such as brainstorming or voting. Whilst co-ordination referred as complex interdependent work [Mah95]. Collaborative management tools facilitate and manage group activities such as (i) collaborate and share structured data and information, (ii) interact and share with clients in a private online environment, and (iii) organize social relations of groups [Cha98]. Software development teams may have difficulties in communication remotely without software development collaboration tools. Groupware is a powerful means for discussing ideas, distributing tasks, monitoring project progress, communicating with developers, and keeping a team in synchronization [Nes19].

2.2 Pull-based Software Development Model

GitHub introduced a fork & pull model for collaboration on distributed software development. Anyone can create a fork of a repository, use code as their own and make changes that they may contribute back through pull requests [Jia+17]. Forking is similar to creating a local copy of the repository. New contributors with no access to the main repository can fork the repository and make code changes. A pull request is then created to tell the maintainers of the main repository that the fork owner has changes that need to be pulled into the main repository. A code review is performed prior to the change being merged into the main repository [She+14]. On GitHub, collaboration is based upon pull requests which allow us to compare the content of two branches. After adding or modifying a feature one can open a pull request, which people can start a discussion about the commits and give feedback [Ern17]. Complex issues require more effort to close and include several commits. Once the commit with the fix is merged into the default branch, the issue will be closed [Pip15]. It works the same way as closing an issue from a commit message does and it works across repositories. It is also possible to solve more than one issue with a single pull request because some issues are related to other issues [GPD14].

The pull-based model is widely adopted by open source projects hosted on the modern collaborative coding platforms such as BitBucket, Gitorius, and GitHub [Yu+15]. On GitHub, different social media features are integrated with the pull-request mechanism. Hence, users can freely watch the development activities of popular repositories, follow distinguished developers and comment on others' contributions [TDH14b]. The socialized pull-request paradigm offers greater conveniences for collaborative development when compared to the traditional development methods (e.g., email-based patching) [Yu+16]. The pull-based development model is used to integrate incoming changes into a project's code-base. Changes to a software project can be proposed by external contributors, developers without sharing access to the central repository with the core team [Vas+15]. Instead, they can create a fork of the central repository, work independently and, whenever ready, request to have their changes merged into the main development line by submitting a pull-request [Yu+15]. Developers can commit changes on a pull request that was created from a fork of their repository with permission from the pull request creator. Developers can only make commits on pull request that: (i) are opened in a repository that developers have push access to and that were created from a fork of that repository; (ii) have permission granted from the pull request creator; and, (iii) do not have restrictions that will prevent developers from committing [RR14].

2.3 Version Control Systems

Software developers operating in groups are regularly writing new code and changing (adding or deleting) existing code. The code of a project, app, or software component is usually organized in an exceedingly folder structure or file tree. One developer in a team may be working on a new feature whereas another developer fixes an unrelated bug by changing existing code, every developer could create their changes in several components of the file tree [HLT09]. Version control systems help teams tracking every individual change by each contributor, and helping prevent conflicting from concurrent work [AB02]. Changes created in a part of the software are often incompatible with those created by another developer working at the same time. This problem ought to be discovered and solved in an orderly manner while not blocking the work of other members of the team [GHM98].

Version control systems have seen great enhancements over the past few decades. There are plenty of version control systems available such as Git, Beanstalk, Perforce, Mercurial. There are three types of version control systems: (i) local version control system which keeps track of all files existing in local system, (ii) centralized version control systems which is a centralized server that keeps track of all changes in files, (iii) distributed version control system of which all information is distributed, and if a server dies then client repositories can help restoring server data [Kum18]. Distributed version control systems allow full access to every file, branch, and iteration of a project, and allow every user access to a full and self-contained history of all changes [Han19]. Nowadays, developers are also using GitHub as a coordination channel. To offer a web-based service for software development, GitHub was created in 2008. Particularly, it takes advantage of the Git version control system. It makes easier for a team of developers coding together and thus efficiently handle large projects [Per+16]. GitHub is useful for project managers and web teams that are looking into ways to improve their interaction and work on different parts of the project.

In GitHub, developers are allowed to make changes, adapt and improve software from their repositories. Each repository contains all project files, as well as, the revision history of each file[RC16]. Several developers can collaborate on the same repository which can be either public or private. For collaboration purposes, GitHub provides a web interface to the Git repository and management tools, which also supports social coding. Software developers may think GitHub as a site for social networking in which they can receive updates for specific projects, communicate with other developers, follow each other, and rate each other's work[RC16]. In GitHub, there are three important terms which are often used by developers: fork, pull request, and merge. A fork is simply a repository that has been copied from the account of a member to the account of another member. Forks allow a developer to make modifications without affecting the original code [Jia+17]. While pull request allows developers to inform others about the changes pushed into the repository. If the developer would like to share her code changes, she should send a pull request to the owner of the original repository. GitHub's merge feature lets developers integrate modifications into the repository [Kal+16]. If, after reviewing the code changes, the original owner wants to have it into the repository, she can accept the modifications and merge them with the original repository.

2.4 Developers Communication on Software Development

Communication tools support collaborative software development. Initially, the available tools were mostly asynchronous forms of communication such as mailing lists or relying on periodical publications to spread information [Thi+07]. Synchronous and economical communication was restricted to telephone calls. In general, software development requires a great deal of information exchange. Robillard and others [RR00] have shown that most of a developer's time is spent on communication activities. Communication can be hindered by several barriers, such as socio-cultural, linguistic, knowledge, geographical and temporal barriers [KR19].

Research on communication in collaborative software development is conducted to improve the understanding of the implications of different communication methods on the success of the development process and the final product. In a collaborative environment, communication can be achieved either by synchronous media or asynchronous media. Email is an asynchronous method of exchanging digital messages between developers using digital devices such as computers, mobile phones and other electronics [Nii11]. Instant messaging (IM) systems popped out during the 90's and there are lots of provides for different systems nowadays. In most companies at least one IM system is in use - some even have many of them which makes it even more difficult to do work without interruptions [CI14]. The problem with IM is that sometimes people miss the information as they are not at the time on computer. Wiki systems are another communication channel that is widely used in many organizations. While wiki systems can be used powerfully to provide information to lots of people, they usually lack the functionality to notify people about new information [Lan+10].

While synchronous technologies are audio and video conference, used for communication between developers in real-time such as Zoom, GoToMeeting and Highfive [NPL09]. Chat systems are good for sharing information for a group of people at the same time. Also, they do not overload people as much as IM systems because the information can be read when the person has time and can concentrate on the information. The problem with these systems is that they are really hard to get back to previous conversations. Moreover, they do not provide good support with new content whether it is good or bad, due to lack of notifications. Many social media platforms, such as Facebook (Workplace), offer tools for companies to have their own, internal, social media channels for their employees [Lan+10]. This is an interesting way of communication in businesses. Drawing to a whiteboard or a paper can open new perspectives to the problem, the main issue with drawing is that it can only be distributed to the people in the same room. Besides, most of the time drawing requires also some verbal communication to make it understandable.

2.5 Developers Coordination on Software Development

To carry out organizational tasks successfully, an appropriate combination of organizational structure, processes, communication, and coordination mechanisms is required [Cat+07]. The coordination problems encountered in software development projects depend on an intricate relationship of several factors. First, elements that influence how closely-coupled the work is, such as complexity and uncertainty of the interfaces, as well as, whether the work is carried out sequentially or concurrently [Cat+07]. Secondly, factors that influence the ability to communicate and coordinate, such as geographic location, whether the communication is direct or through an intermediary, and the quality of documentation play a vital role. Moreover, there are some organizational factors that should also be considered significant mediators. These mediators include processes, structure, and goal alignment in the organization. Coordination helps in integrating developers' work which turned into a group product. In an organization, technical teams follow a linkage process which may contain different team design principles and communication to coordinate their work [KWC94]. Therefore, the scope of software engineering projects is increased with the advances and enhancement in IT and other formal methodologies used for coordination. Before, software development teams used to be very small in numbers and less disperse, but today software development teams are generally large, diverse, better trained, and more distributed in nature [HM19]. Furthermore, their tasks are more advanced, complex, uncertain, and more fluid compared to the past - all this despite enhancements in hardware and software that have changed individual work and coordination less burdensome.

2.6 Communication Model on GitHub

As mentioned, communication is a key value in software development transmitting information between developers. The need to communicate effectively pervades software development, operations, and support [Thi+07]. In an organization, developers may use GitHub to improve their communication. In the pull-based software development model, each developer work in their fork of the repository. This allows developers to create whatever branches they want without polluting the main repository. Before this, we had dozens of branches sitting in the main repository that belonged to no one. Now, each developer manages their branches and does not confuse project newcomers with a pile of old or incomplete branches [Dau16]. Pull requests include the ability to request a review from another developer, and even to enforce rules that require a review before a pull request may be merged. Figure 2.1 presents a sequential process that represents the pull-based software development model which includes six steps as described below.

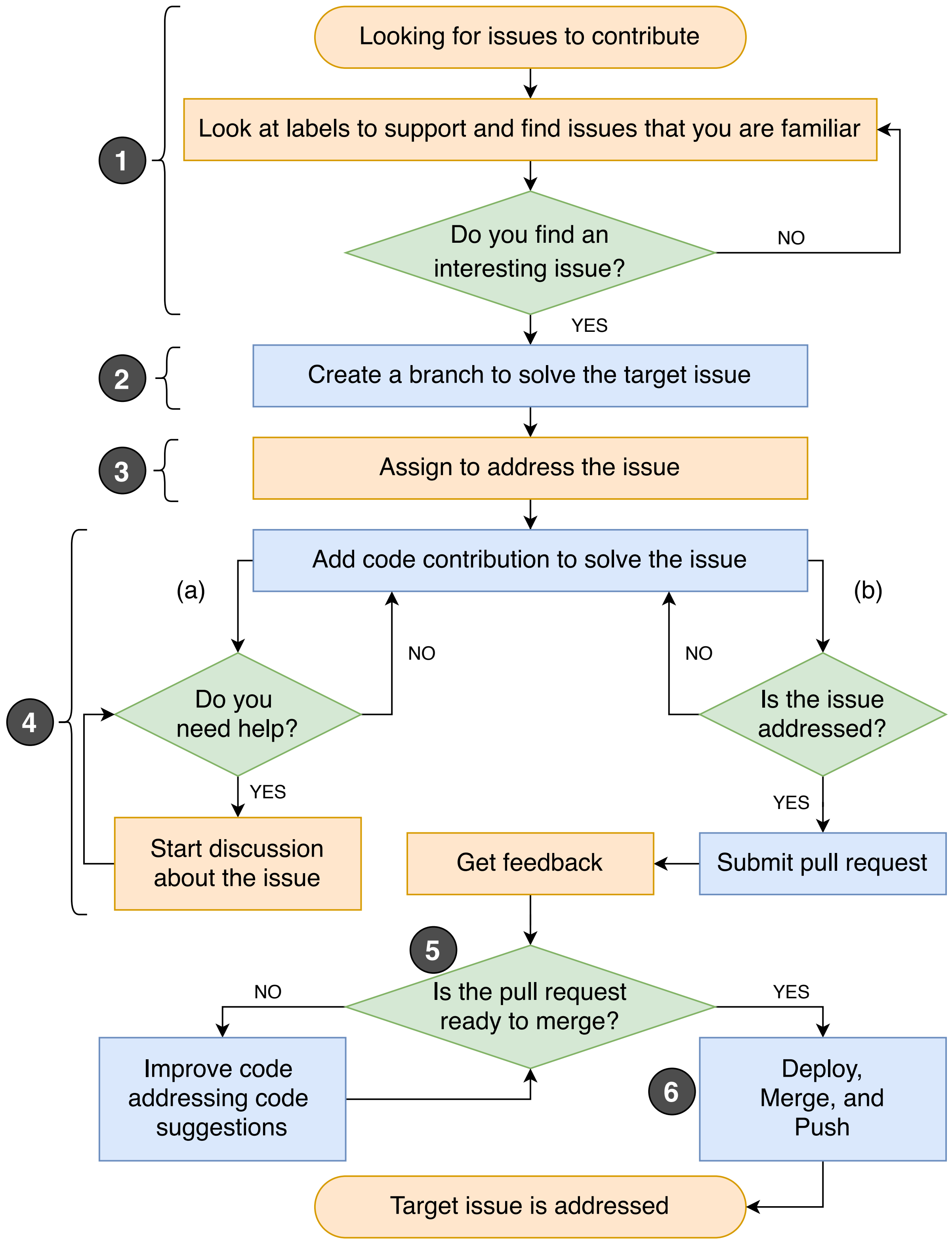

1
Looking for issues to contribute
Look at labels to support and find issues that you are familiar
Do you find an interesting issue?
NO
YES
2
Create a branch to solve the target issue
3
Assign to address the issue
4
Add code contribution to solve the issue
(a)
(b)
NO
NO
Do you need help?
Is the issue addressed?
YES
YES
Start discussion about the issue
Get feedback
Submit pull request
5
Is the pull request ready to merge?
NO
YES
Improve code addressing code suggestions
6
Deploy, Merge, and Push
Target issue is addressed

Figure 2.1: A GitHub workflow

1. **Find an issue to solve and discuss the task (e.g., bug fix and new feature introduction)**

   The first step when contributing to a project is to visit the project and find an issue. Developers can collect user feedback, report software bugs, and organize tasks they would like to accomplish with issues in a repository. Issues are considered as more than just a place to report software bugs. To stay updated on the most recent comments in an issue, one can watch an issue to receive notifications about the latest comments [Bis+13]. With issues, one can:

   - Track and prioritize work using project boards.
   - Associate issues with pull requests.
   - Create issue templates to help contributors open meaningful issues.
   - Transfer open issues to other repositories.
   - Pin important issues to make them easier to find, preventing duplicated issues, and reducing noise.
   - Track duplicate issues using saved replies.
   - Report comments that violate GitHub's Community Guidelines.

   Developers may also look for labels to find issues of their interest. Each label has a name and color which can be assigned to issues to prioritize and categorize them.

2. **Create a new branch (or fork) to address the target issue.**

   When beginning work on an issue locally, the first thing developers need to do is to create a branch for that piece of work. Distributed version control systems enable development in a more independent manner. Each member of a team uses branches to have their clones of a project and to work on tasks in parallel with other members for minimizing the touchpoints between them [Kal+14a]. By moving to a peer-to-peer configuration, distributed version control systems offer an alternative code management model, leading to their workflows. Potential contributors do not write directly to the main repository of a project. Rather than, these contributors make their changes independent of each other by creating a fork (clone) of the repository. Hence, a fork is a copy of a repository that allows you to freely experiment with changes without affecting the original project [Mer15].

3. **Assign developers to address the proposed issue**

   The next step is to assign an issue to work on depending on suitable skill set. On GitHub, the assignment of issues to developers is considered as part of a work-flow functionality for solving issues. Research on bugs or defects in open source projects has been abundant [RK10][Sin13]. However, most of this research focused on assigning specific bugs to the right developers ("bug triage"), rather than using issue closure as a development process performance measure. Triaging bug reports is a labor-intensive and time-consuming task that assigning incoming bug reports to appropriate bug fixers [MKN09]. There is plenty of research based on machine learning and information retrieval techniques to triage incoming bug reports practices of open source development.

4. **Solve the issue by code contribution**

   After creating a branch, developers solve issues locally on their forked branch. Developers can post their comments to discuss and manage the issue resolution. They can comment on the solution on the opened issue to get help, if necessary, and give an opportunity to other developers to provide feedback. Normally, there are two types of comments [Kal+14b]:

- Discussion: Comments on the overall content of the pull request. Interested parties participate in technical discussions concerning the suitability of the pull request altogether.

- Code Review: Comments on specific sections of the code. The reviewer makes notes on the commit *diff* and *pinpoint* potential improvements.

5. **Create a pull-request to integrate their solution referring to the issue they are solving.**

   After discussion and code review, developers can change the source code accordingly and tell others by initiating a pull request. Pull requests are tightly integrated with the underlying repository. Maintainers accept the pull request and can see exactly what changes are going to be merged. The issue part of the pull request is used to keep track of the discussion comments. Developers are encouraged to refer to issues or pull requests in commit messages or in issue comments. GitHub extracts such references and presents them as part of the discussion flow [Kal+14b]. When a set of changes is ready to be submitted to the main repository, developers create a pull request, which specifies a local branch to be merged with a branch in the main repository. A member of the project's core team is then responsible to inspect the changes and pull them to the project's master branch. If changes are considered unsatisfactory (e.g., as a result of a code review), more changes may be requested; in that case, contributors need to update their local branches with new commits [GPD14].

6. **Deploy, merge, and push to address the target issue.**

   In GitHub, a developer can deploy from a branch for final testing in production before merging to the master branch. Once a pull request has been reviewed and the branch passes a benchmark of tests, it is time to deploy the changes to verify them in production. If the branch causes trouble, the developer can roll it back by deploying the existing master into production. After verifying the changes in production, the code should be merged into the master branch. Once merged, pull requests preserve a record of the historical changes to the code. Because they are searchable and allows anyone to go back in time to understand why and how a decision was made. Using some keywords into the text of a pull request can associate GitHub issues with the code changes. Hence, when the pull request is merged the related issue should also be closed.

2.7 Relation among GitHub features

GitHub is considered more than an issue tracker. It represents a social network in which technical assets are also linked to the platform. Its functionalities are used for discussing different kinds of details such as bugs, and new features. On GitHub, issues are a great way to keep track of tasks, enhancements, and bugs in a project. Developers can discuss and review the potential changes, ask for code reviews, and feedback through pull requests in a projects. Pull requests can be used to close one or more issues by using a reference keyword. A commit, or revision, is a personal amendment to a file (or set of files). It is similar to saving a file, except that every time we save Git creates a unique identifier (i.e. the commit hash) that allows us to keep track of what changes were made, when, and by who. Commits usually contain a message which is a brief description of what changes were made. Labels help us to organize issues. Developers can apply labels to issues and pull requests to indicate priority or category. In repositories, a developer with write access, can create, assign, edit (name, color, description), or delete labels. Labels are consistent across repositories, and make easier switching among projects.

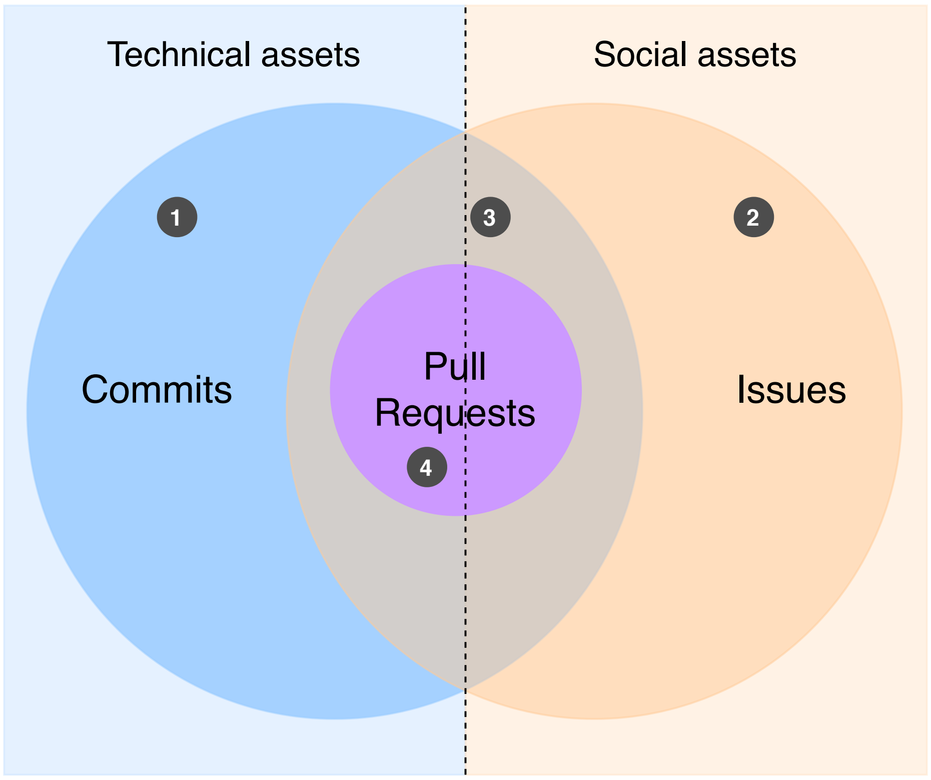


Figure 2.2: The relation between issues, commits and pull requests

In Figure 2.2, we present relations among issues, commits and pull requests on GitHub. In GitHub pull requests is a subset of issues presented as set 4. It means, every pull request is an issue, but not all issues are pull requests. Developers create issues for describing different tasks and propose solutions for them through pull requests. With a pull request one can solve more than an issue, if the issues are related. Commits are either related to the deletion or addition in the code. Developers link commits with related issues presented as set 3. Thus, a single commit may help to solve multiple issues. Commits are technical assets while issues represent social assets. As pull requests are related to commits and may also retrieve communication among developers, we consider pull requests a hybrid asset.

3 Related Work

In this chapter, we present different studies on how researchers have investigated coordination in collaborative software development. Furthermore, we provide details of other studies that have been conducted on coordination for software development.

Communication channels play an essential role in supporting coordination activities within a community of practice [Sto+14]. Various researchers have investigated developers' collaboration through communication channels and tools, such as mailing lists, IRC chat logs, issue trackers, and social networks (e.g., GitHub and Stack Overflow). For instance, Bird et al. [Bir+08] investigated the relationship between communication structure and code modularity. They found a relationship between communication and code collaboration behavior for sub-communities. Guzzi et al. [Guz+13] analyzed a large sample of e-mail threads from Apache Lucene's development mailing list. They found that developers participate in less than 75% of the threads, and in only about 35% of the threads source-code details are discussed. LaToza et al. [LVD06] interviewed eleven developers to discover common practices in software development. They found several barriers preventing e-mail usage, highlighted advantages of face-to-face communication, and that the use of more interactive communication channels is more desirable than e-mails. Panichella et al. [Pan+14] analyzed three communication channels (mailing lists, issue trackers, and IRCs) and code changes of seven projects. They found that not all developers use all communication channels, and socio-technical relationships may change when using different communication channels and tools. Dabbish et. al [Dab+12] examined how individuals interpreted and made use of information about others' actions on code in an open social software repository. They found that four key features of visible feedback drove a rich set of inferences around commitment, work quality, community significance, and personal relevance. They believe that collaboration, learning, and reputation management are supported by these inferences within community.

In an extensive study, Storey et al. [Sto+14] mapped different communication tools, such as e-mail lists, IRCs, SourceForge, GitHub, and Stack Overflow. They hypothesized that knowledge in software engineering is embedded in: (i) people's heads, (ii) project artifacts, (iii) community resources, such as forums, blogs, and discussion groups, and, (iv) social networks. According to their study, GitHub is the only tool able to represent (the last) three types of knowledge. Nakakoji et. al [NYY10] identified two distinct types of communication in software development, coordination communication, and expertise communication. They argued that different sets of design guidelines are necessary for supporting each type of communication. They also described nine design guidelines to support expertise communication based on the theories of social capital and models of supporting collective creativity. Hattori et. al [HL10] built Syde, a tool infrastructure to re-establish team awareness by sharing change and conflict information across developers' workspaces. The novelty of their approach is that they model source code changes as first-class entities to record the detailed evolution of a multi-developer project. Therefore, precisely changed and amendment information is sent to interested developers by Syde. Gutwin et al. [GPS04] looked at requirements and mechanisms for collaborative development on three open source projects. They found that distributed developers maintain both a general awareness of the entire team and more detailed knowledge of people that they plan to work with. They observed that mailing lists and chat tools were primary channels for maintaining awareness, coordination, and communication. In an empirical study, Seaman et al. [SB97], showed that developers take less time to communicate when they are familiar with one another and when they work in close physical proximity. Furthermore, an additional effort is required to communicate when certain mixtures of organizationally close and distant participants interact. The results provide a better understanding of how organizational structure helps or hinders communication in software development.

The popularity of the pull-based development model and GitHub has increased the interest of researchers. For example, Singer et al. [Sin+13] and Dabbish et al. [Dab+12] explored the value of social mechanisms in GitHub. Both studies found that transparency helps developers connect, collaborate, create communities, share knowledge, and discover new technologies. Tsay et al. [TDH14a] analyzed the association of various technical and social measures with the likelihood of contribution acceptance. They found that pull request acceptance is related to (i) the strength of the social connection between the submitter and the project manager, (ii) the submitter's prior interaction, (iii) the number of comments, and (iv) the current stage of the project. Gousios et al. [GPD14] analyzed millions of pull requests to study the effectiveness and efficiency of contributors handling pull requests. They discovered that the time to merge a pull request is influenced by the developer's previous track record, the size of the project and its test coverage, and the project's openness to external contributions. Liu et al. [LLH16] conducted a quantitative study on the specific effects of pull requests on the project. They found that pull requests help to increase the social impact, resulting in larger development activity.

Different coordination tools and channels are used by software developers to accomplish their project work. The diversity of these tools has dramatically increased over the past decade, giving rise to a wide range of socially enabled communication channels and social media that developers use to support their activities. The availability of such social tools is leading to a participatory culture of software development, where developers want to engage with, learn from, and co-create software with others. In an evolutionary study on coordination channels, Storey et. al [Sto+17] described which channels developers find essential to their work and gain an understanding of the challenges they face using them. Their findings lay the empirical foundation for providing recommendations to developers and tool designers on how to use and improve tools for developers. Furthermore, Tien-Duy et al. [Le+15] explored links between issue reports and their corresponding commits. They used ChangeScribe approach which analyzes code changes and creates commit messages by combining several code summarization techniques. Their approach used a discriminative model which is used to predict if a link exists between a commit message and a bug report. It is known as RCLinker, which stands for Rich Context Linker. They have also presented a comparison of RCLinker against MLink, which is the latest state-of-the-art bug linking approach.

Despite the number of studies investigating the coordination among contributors, we did not find any study exploring how contributors in fact link technical with social assets in the pull-based development model in GitHub. Previously, several studies [Bir+08] [Guz+13] [LVD06] [Pan+14] [Sto+14] [GPS04] have explored communication channels and tools, such as mailing lists, IRC chat logs, issue trackers, and social networks. The pull-based development model and GitHub have already been investigated and used by various studies [Sin+13], [Dab+12], [TDH14a], [GPD14], and [LLH16] showing the importance of GitHub across different communication channels. Furthermore, few studies [NYY10] [SB97] have investigated how organizational structure supports communication in software development. More similar to our study, Tien-Duy et al. [Le+15] provide a strategy to link social and technical assets. In other words, they have investigated links between bug reports (social assets) and source code commits (technical assets) via comparison of textual information in commit messages with textual contents in the bug reports. Our study, on the other hand, does not provide a strategy, but check how developers do that link of technical and social assets on GitHub. Hence, we intend to empirically understand how developers relate technical and social assets, and by means of a survey to find out why they do not link these assets in all cases.

4 Methodology

In this chapter, we present details of our study design. In Section 4.1, we present our research questions. In Section 4.2, we show subject projects as well as the process to select them. In Section 4.3, we provide details about variables that we use to fetch data from repositories. Finally in Section 4.4, we show how we operationalize the answer of our research questions.

4.1 Research Questions

GitHub provides several features to support coordination among developers such as issues, pull requests, commits, and labels. Issues are a great way to keep track of tasks, enhancements, and bugs in a project. Pull requests represent what is going to change in the code or project. Commits are individual changes to a file (or set of files). Labels are short-terms to describe the type of an issue.

Linking social and technical assets is important for measuring the quality of various parts of a software system, predicting defects, improve recommendations of developers for solving specific bugs, prioritize issues to be solved, and improve software coordination [Le+15]. In our study, we investigate three relationships among GitHub features. First, we investigate whether developers link problem descriptions (issues) with the solution of these problems (pull requests). To get a deeper understanding, we investigate this relationship among issues and pull requests in both directions. Our first research question summarize this investigation.

RQ1: Do developers link pull requests with related issues?

Second, we investigate whether developers link commits with the problems their described (issues) and addressed (pull requests). Again, we investigate this relationship in both directions. We summarize this second analysis in the following research question.

RQ2: Do developers link commits with issues?

Last, we investigate whether developers assign labels to the problems (issues) and solutions (pull requests). Labeling allows developers to quickly filter and find issues they are looking for. To get more profound knowledge, we investigate the relationship between labels with issues and pull requests. The following research question abstracts this investigation.

RQ3: Do developers effectively use labels to represent the type of issues?

4.2 Subject Projects

We use GitHub repositories to collect data that helps in supporting metrics and providing statistical results. GitHub is one of the largest coding communities and can provide wide exposure for our study and it has been investigated and used by several studies [Dab+12], [Kal+14b], [LLH16], [TDH14a], [Yu+15].

Initially, we retrieved 100 most popular repositories on GitHub, as determined by the number of stars [BV18]. Then, we applied the following five filters: (i) no programming projects: it excludes projects that do not have a programming language classified as the main language (i.e., the main file extension), (ii) less than 50 issues and 50 pull requests: it removes repositories which have less than 50 issues and 50 pull requests, (iii) inactive: it eliminates projects which have less than two commits per month in the last six months, (iv) not possible to rebuild most merge scenarios: it discards projects that were not possible to rebuild most of the merge scenarios, and (v) balancing programming language: it excludes JavaScript projects until they represent less than half of the systems in our dataset of subject systems. Most projects are JavaScript projects, hence, we removed the last popular projects to not let this programming language dominate our dataset. Figure 4.1 illustrates how these filters are applied to GitHub repositories and presents the number of projects remaining after each filter.

In Table 4.1, we present the name of each repository, programming language, the number of issues, pull requests, commits, labels, and stars, and which filter excluded a target project. Hence, if the *Filter* column is empty we selected the project.

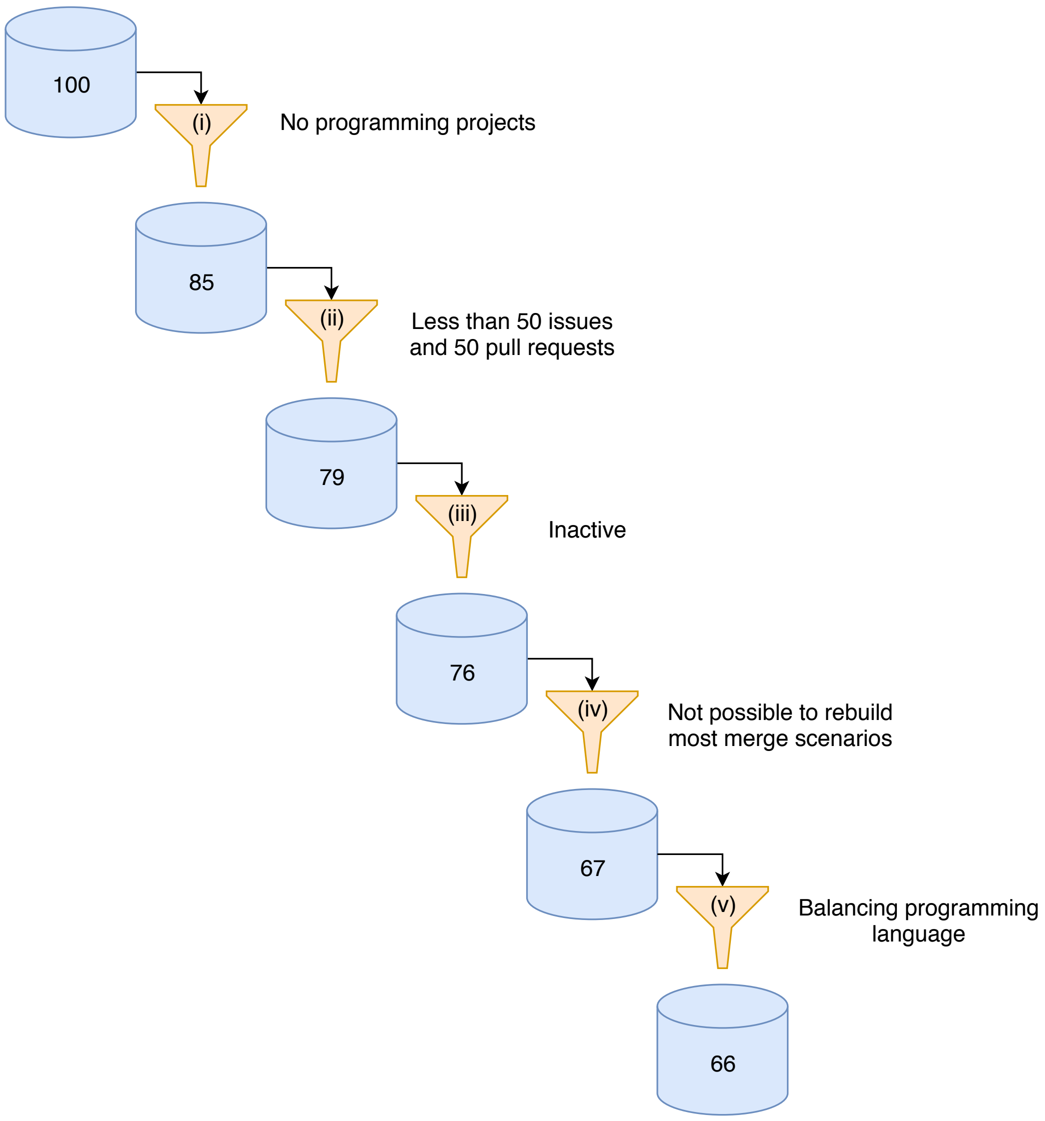

100
(i)
No programming projects
85
(ii)
Less than 50 issues and 50 pull requests
79
(iii)
Inactive
76
(iv)
Not possible to rebuild most merge scenarios
67
(v)
Balancing programming language
66

Figure 4.1: Filters applied to projects

Table 4.1: Overview of the subject projects

Id	**Project name**	**Program. language**	**#Stars**	**#Issues**	**#PR**	**#Com**	**#Labels**	**Filters**
1	freeCodeCamp	JavaScript	306270	14017	23314	25050	55	
2	996.ICU	Rust	247602	-	1806	3019	14	(ii)
3	vue	JavaScript	151206	8537	1662	3059	33	
4	react	JavaScript	138635	8293	8767	12566	50	
5	tensorflow	C++	137425	21390	12780	72086	73	(iv)
6	bootstrap	JavaScript	136882	18895	10257	19135	31	
7	free-programming-books	No language	135253	407	3012	5170	13	(i)
8	awesome	No language	123542	239	1420	885	8	(i)
9	You-Dont-Know-JS	No language	111462	793	747	1563	11	(i)
10	oh-my-zsh	Shell	97706	3058	5270	5349	36	
11	coding-interview-university	No language	96626	104	364	1301	10	(i)
12	gitignore	No language	93819	-	3242	3194	4	(i)
13	developer-roadmap	No language	93416	401	130	241	7	(i)
14	javascript	JavaScript	90045	936	1178	1783	16	
15	d3	JavaScript	88250	2018	1088	4195	15	
16	vscode	TypeScript	86754	77805	6268	57427	242	(iv)
17	linux	C	82807	-	721	873287	-	(ii)
18	react-native	JavaScript	82665	18125	8921	18744	166	
19	CS-Notes	Java	81607	338	445	3500	11	
20	flutter	Dart	78164	27637	16145	16358	206	
21	electron	C++	78136	11763	8952	22704	136	
22	system-design-primer	Python	75627	95	237	274	16	
23	awesome-python	Python	74959	158	1259	1392	8	
24	create-react-app	JavaScript	72998	5046	2815	2393	29	
25	the-art-of-command-line	No language	67171	154	496	1185	14	(i)
26	axios	JavaScript	66063	1942	562	883	24	
27	go	Go	65878	34626	894	42061	65	
28	node	JavaScript	65737	10738	19340	28602	162	
29	public-apis	Python	64243	114	981	2128	4	
30	anima-te.css	CSS	62802	663	322	454	15	

31	free-programming-books-zh_CN	No language	62345	342	395	887	7	(i)
32	Font-Awesome	JavaScript	61276	15091	584	61	73	
33	Java-Guide	Java	61006	233	316	1889	12	(i)
34	Python	Python	60627	192	1325	1289	19	
35	kubernetes	Go	60102	32664	52699	85866	187	(iv)
36	angular.js	JavaScript	59614	8954	7887	8967	97	
37	models	Python	58850	4741	2869	3925	21	
38	build-your-own-x	No language	58097	294	127	375	-	(i)
39	youtube-dl	Python	57223	18281	3565	17361	39	
40	three.js	JavaScript	56415	9241	8676	30606	20	(iv)
41	javascript-algorithms	JavaScript	56263	115	310	789	9	(iii)
42	puppeteer	JavaScript	55904	3458	1693	1602	13	(iii)
43	laravel	PHP	55890	-	3492	6106	8	(ii)
44	moby	Go	55531	20075	20366	38010	126	(iv)
45	TypeScript	TypeScript	55219	23840	10258	29209	112	
46	computer-science	No language	53689	407	222	739	8	(i)
47	angular	TypeScript	53477	19119	14198	15923	137	
48	ant-design	TypeScript	52997	14354	5070	16275	119	
49	jquery	JavaScript	52414	1933	2548	6438	39	
50	java-design-patterns	Java	52288	509	532	2286	28	
51	material-ui	JavaScript	51753	9700	8366	10436	131	
52	webpack	JavaScript	51632	6683	3135	9774	60	
53	awesome-vue	No language	51382	157	2941	2810	8	(i)
54	redux	JavaScript	51131	1699	1888	2911	13	
55	30-seconds-of-code	JavaScript	50913	207	797	4643	19	
56	atom	JavaScript	50266	15165	4585	37289	108	
57	thefuck	Python	50003	501	493	1542	14	
58	swift	C++	49803	-	28188	97163	16	(ii)
59	awesome-go	Go	49076	201	2599	3232	16	
60	reveal.js	JavaScript	48957	1713	810	2341	12	
61	socket.io	JavaScript	47885	2761	731	1720	24	
62	flask	Python	47258	1808	1589	3774	16	
63	Semantic-UI	JavaScript	46686	6027	856	6684	25	
64	lantern	Go	46131	-	878	6101	45	(ii)
65	express	JavaScript	45968	3165	918	5558	36	
66	Chart.js	JavaScript	45880	4802	1821	2816	16	

67	elastic-search	Java	45422	21729	27366	49225	395	(iv)
68	keras	Python	45091	9664	3832	5341	26	
69	django	Python	44945	-	11976	27496	6	(ii)
70	rails	Ruby	44510	13160	24446	75043	41	(iv)
71	shadow-socks-windows	C#	44155	2285	333	1266	17	
72	httpie	Python	43609	559	236	1112	16	
73	html5-boilerplate	JavaScript	43555	1151	996	1760	18	
74	json-server	JavaScript	43538	816	221	779	8	
75	spring-boot	Java	43389	15145	3803	23972	34	(iv)
76	Front-end-Developer-Interview-Questions	HTML	42809	97	446	782	20	(i)
77	storybook	JavaScript	42703	4232	4401	22327	89	
78	moment	JavaScript	42649	3478	1788	3724	35	
79	resume.github.com	JavaScript	42627	83	173	263	6	(i)
80	awesome-machine-learning	Python	42359	61	578	1323	7	
81	netdata	C	42090	4214	3010	9143	48	
82	lodash	JavaScript	42057	3508	1015	7988	14	
83	next.js	JavaScript	42045	5102	4082	4432	21	
84	element	Vue	41924	13163	4112	4349	78	
85	markdown-here	JavaScript	41581	529	54	807	15	
86	meteor	JavaScript	41385	8146	2584	21976	72	
87	bitcoin	C++	41141	5058	12170	22103	44	(iv)
88	RxJava	Java	40876	2794	3268	5589	34	
89	requests	Python	40651	3005	1991	5921	23	
90	every-programmer-should-know	No language	40535	28	121	231	3	(i)
91	rust	Rust	40521	31969	34458	102430	264	
92	ansible	Python	40384	25302	37497	48264	351	
93	redis	C	39414	3844	2626	8511	40	
94	ionic	TypeScript	39406	16359	3344	9919	22	
95	material-design-icons	CSS	39127	837	98	124	11	
96	jekyll	Ruby	38992	4030	3709	10666	82	
97	react-router	JavaScript	38331	5192	1828	4529	8	
98	yarn	JavaScript	37183	5401	2142	2309	24	
99	materialize	JavaScript	36674	5052	1397	3871	47	(v)
100	node-v0.x-archive	No language	35316	6383	3798	2	103	(iii)

Id: Identifier, **PR**: Pull requests, **Com**: Commits

4.3 Data Acquisition

Given that software development is social in nature, we build socio-technical relationships to obtain an authentic representation of developers' contributions and GitHub activity. Furthermore, we detail software properties we measure to retrieve developers' contributions and GitHub activity, as we detail next.

Code contributions: Our strategy for contribution data acquisition consists of five steps. First, we clone a subject project's repository. Second, as merge commits can be identified in Git when the number of parent commits is greater than one, we identify merge scenarios by filtering commits with multiple parent commits. Third, for each merge commit, we retrieve a common ancestor for both parent commits (i.e., the base commit). Fourth, we (re)merge the parent commits and retrieve the measures for the metrics by comparing the changes that occurred since the base commit until the merge commit. Finally, we store all data and repeat steps 3 to 5 for each merge scenario found in step 2. We preferred to use the data of commits that have a context instead of using random commits that belong to the project.

GitHub activity: Considering experience from previous work presented in Chapter 2 and the benefits, comprehensiveness, and popularity of GitHub when compared to other communication tools and channels, we chose to rely on our study on the GitHub platform. Another benefit is that by using GitHub, projects normally follow the pull-based development model which includes three of the four features investigated in this study (issues, pull requests, and labels). Aware of the drawbacks of using only one communication channel, we select only projects that extensively use the GitHub communication mechanism (see filters (ii) and (iii) of Section 4.2). Given a limitation in the number of calls we can request on the GitHub API, we had to request tokens from several colleagues that have a GitHub account. To collect data, we required only three scopes (one for repository access and two for user access). The scopes are presented as follows:

1. public_repo: Access public repository
2. read:user: Read all user profile data
3. user:email: Access user email addresses (read-only)

Investigated Variables: Aiming at providing different perspectives of our data, we computed a set of 16 variables. Table 4.2 describes each one of these variables. We use them to compose equations and answer research questions, as described in Section 4.4.

Table 4.2: List of variables of metrics

Variable name	**Description**
#issues	Counts the total number of issues
#issues (not pull requests)	Counts the number of issues that are not pull requests
#pull requests	Counts the total number of pull requests
#issues related with issues	Counts the number of issues which are related to other issues
#issues (not pull requests) linked with pull requests	Counts the number of links between issues that are not pull requests with pull requests
#pull requests linked with issues	Counts the total number of links between pull requests and issues
#commits	Counts the total number of commits
#commits linked with issues	Counts the number of links between commits and issues
#commits linked with issues (not pull requests)	Counts the number of links between commits and issues that are not pull requests
#commits linked with pull requests only	Counts the number of links between commits and pull requests
#issues linked with commits	Counts the number of links between issues and commits
#issues (not pull requests) linked with commits	Counts the number of links between issues that are not pull requests with commits
#pull requests linked with commits	Counts the number of pull requests linked with commits
#issues linked with labels	Counts the number of links between issues and labels
#issues (not pull requests) linked with labels	Counts the number of links between issues that are not pull requests with labels
#pull requests linked with labels	Counts the number of links between pull requests and labels

4.4 Operationalization

We operationalize our research questions through the variables presented in Table 4.2. To answer RQ1, we measure **the number of links between pull requests and issues**. To answer RQ2, we compute **the number of links between commits with issues and pull requests**. To answer RQ3, we compute **the number of links between labels with issues and pull requests**.

Answering Research Question 1: In a GitHub repository developers may link pull requests with issues in four ways, as we show in Figure 4.2. These ways are: (i) one pull request linked with one issue [1:1], (ii) one pull request linked with multiple issues [1:n], (iii) multiple pull requests are linked with one issue [m:1], (iv) multiple pull requests are linked with multiple issues [m:n]. For some cases, we expect that developers do not link issues with pull requests. For instance, when a developer creates an issue to report a problem when setting the repository up, we only expect that another developer describes how to set up the repository in the commentary of this issue. This way, no pull request is expected. We are interested on investigating three relationships between issues and pull requests. First, a general view of the links among GitHub issues. Second, the percentage of issues which are not pull requests linked with pull requests. Third, the percentage of pull requests linked with issues that are not pull requests. These relationships are described in equations 4.1, 4.2, and 4.3, respectively.

$$\begin{array}{c}\text{\% Issues linked with}\\ \text{other issues}\end{array} = \frac{\text{\#Issues related with issues}}{\text{\#Issues}} \times 100 \qquad (4.1)$$

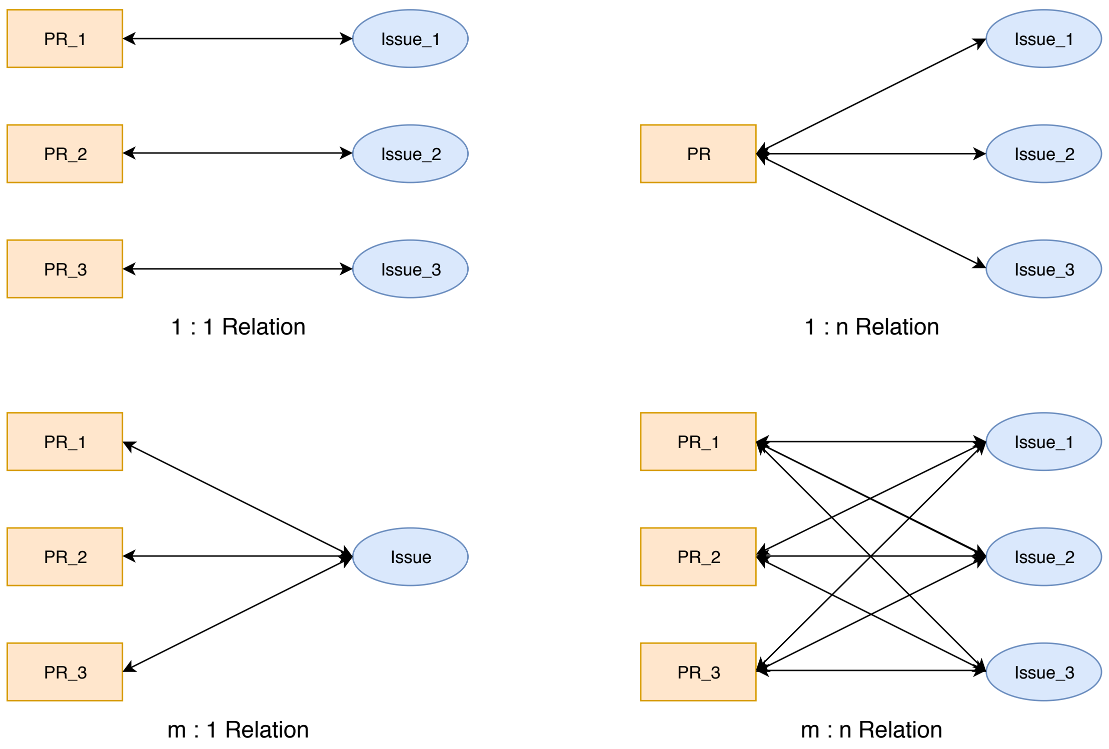


Figure 4.2: Number of ways to link pull requests and issues

$$\begin{array}{c}\text{\% Issues (not pull requests)}\\ \text{linked with pull requests}\end{array} = \frac{\begin{array}{c}\text{\#Issues (not pull requests)}\\ \text{linked with pull requests}\end{array}}{\text{\#Issues (not pull requests)}} \times 100 \tag{4.2}$$

$$\begin{array}{c}\text{\% Pull requests linked with}\\ \text{issues (not pull requests)}\end{array} = \frac{\text{\#Pull requests linked with issues}}{\text{\#Pull requests}} \times 100 \tag{4.3}$$

Answering Research Question 2: We are interested in investigating a total of six relationships among commits with issues and pull requests. First, the percentage of commits linked with issues described in Equation 4.4. Second, the percentage of commits linked with issues that are not pull requests described in Equation 4.5. Third, the percentage of links between commits and pull requests presented in Equation 4.6. Fourth, the percentage of issues linked with commits formulated in Equation 4.7. Fifth, the percentage of issues that are not pull requests linked with commits as shown in Equation 4.8. Last, the percentage of links between pull requests and commits described in Equation 4.9. These equations are important to investigate whether developers link commits with the problems they described (issues) and addressed (pull requests).

$$\begin{array}{c}\text{\% Commits linked}\\\text{with issues}\end{array} = \frac{\text{\#Commits linked with issues}}{\text{\#Commits}} \times 100 \tag{4.4}$$

$$\begin{array}{c}\text{\% Commits linked with issues}\\\text{(not pull requests)}\end{array} = \frac{\begin{array}{c}\text{\#Commits linked with issues}\\\text{(not pull requests)}\end{array}}{\text{\#Commits}} \times 100 \tag{4.5}$$

$$\begin{array}{c}\text{\% Commits linked with}\\\text{pull requests only}\end{array} = \frac{\begin{array}{c}\text{\#Commits linked with}\\\text{pull requests only}\end{array}}{\text{\#Commits}} \times 100 \tag{4.6}$$

$$\begin{array}{c}\text{\% Issues linked}\\\text{with commits}\end{array} = \frac{\text{\#Issues linked with commits}}{\text{\#Issues}} \times 100 \tag{4.7}$$

$$\begin{array}{c}\text{\% Issues (not pull requests)}\\\text{linked with commits}\end{array} = \frac{\begin{array}{c}\text{\#Issues (not pull requests)}\\\text{linked with commits}\end{array}}{\text{\#Issues (not pull requests)}} \times 100 \tag{4.8}$$

$$\begin{array}{c}\text{\% Pull requests linked}\\ \text{with commits}\end{array} = \frac{\text{\#Pull requests linked with commits}}{\text{\#Pull requests}} \times 100 \tag{4.9}$$

Answering Research Question 3: We are also interested on investigating three relationships among labels with issues and pull requests. First, the percentage of links between issues and labels presented in Equation 4.10. Second, the percentage of issues that are not pull requests linked with labels described in Equation 4.11. Third, the percentage of pull requests linked with labels formulated in Equation 4.12. With these equations, we investigate whether developers refer labels with the problems (issues) and solutions (pull requests).

$$\begin{array}{c}\text{\% Issues linked}\\ \text{with labels}\end{array} = \frac{\text{\#Issues linked with labels}}{\text{\#Issues}} \times 100 \tag{4.10}$$

$$\begin{array}{c}\text{\% Issues (not pull requests)}\\ \text{linked with labels}\end{array} = \frac{\begin{array}{c}\text{\#Issues (not pull requests)}\\ \text{linked with labels}\end{array}}{\text{\#Issues (not pull requests)}} \times 100 \tag{4.11}$$

$$\begin{array}{c}\text{\% Pull requests linked}\\ \text{with labels}\end{array} = \frac{\text{\#Pull requests linked with labels}}{\text{\#Pull requests}} \times 100 \tag{4.12}$$

5 Results

In this chapter, we present an overview of our data in Section 5.1. Then, in sections 5.2, 5.3, and 5.4, we answer RQ1, RQ2, and RQ3, respectively.

5.1 Data Overview

In Table 5.1, we present the name of variables (see Section 4.3 for details) and their values collected from 66 subject projects (see Section 4.2). We found that there is a total of 693 292 issues, 284 873 of them are pull requests, and the remaining 408 419 issues are not pull requests. We also collected the total number of commits and labels that are 934 070 and 4 696, respectively. The average number of issues, pull requests, commits, and labels is 10 504, 4 316, 14 152, and 71 per subject project, respectively.

Table 5.1: Measures for variables of metrics

Variable name	**Variable value**
# issues	693 292
# issues (not pull requests)	408 419
# pull requests	284 873
# commits	934 070
# labels	4 696

5.2 Answer of RQ1

RQ1: Do developers link pull requests with related issues?

In Equation 4.1, we calculated the percentage of issues that are linked with other issues, as represented in Figure 5.1a. There is a total of 171 077 issues that are linked with other issues and the total number of issues is 693 292. It represents that 24.67% of the issues are linked with other issues, as presented in Table 5.2.

In Equation 4.2, we calculated the percentage of issues that are not pull requests linked with pull requests, as represented in Figure 5.1b. There is a total of 408 419 issues that are not pull requests and only 27 548 of them are linked with pull requests. It represents that 6.74% of the issues (not pull requests) are linked with pull requests, as presented in Table 5.2. The percentage is very small because developers do not know the solution when they describe the problem. They can link the problem with the solution after they have the solution. Developers should pay more attention to link issues and pull requests in both directions to increase coordination among developers.

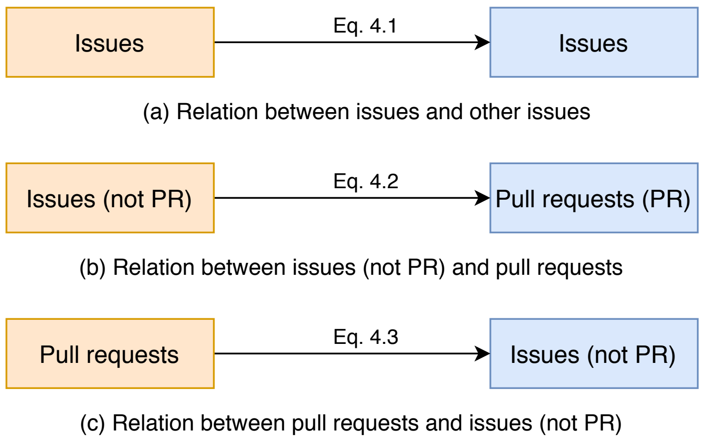

Issues
Eq. 4.1
Issues
(a) Relation between issues and other issues
Issues (not PR)
Eq. 4.2
Pull requests (PR)
(b) Relation between issues (not PR) and pull requests
Pull requests
Eq. 4.3
Issues (not PR)
(c) Relation between pull requests and issues (not PR)

Figure 5.1: Relation between issues and pull requests

Lastly in Equation 4.3, we calculated the percentage pull requests linked with issues represented in Figure 5.1c. The total number of pull requests is 284 873 and the total number of issues is 693 292, however, only 58 560 pull requests are linked with issues. It means that 20.55% of the pull requests are linked with issues, as presented in Table 5.2. The links among problem descriptions and solutions are very important to improve the quality of the system, despite these developers link only one-fifth of the solution with problem description. We expected more links because when they have a solution they normally should know the problem they are solving.

Researchers have to be aware that most of the links do not come from the natural flow (from problem description to solution), the links come from the opposite flow (i.e., from the solution to the problem description). In addition, to make links explicitly and more intuitive, GitHub or external tool builders have the opportunity to provide these links among issues and pull requests in the order they happened to practitioners looking at the data generated by developers.

Table 5.2: Measures for metrics of research question 1

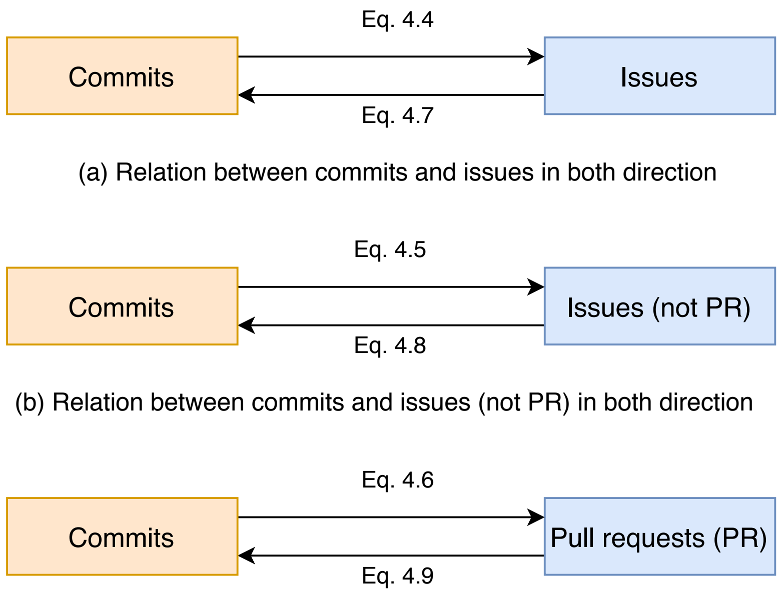
Equation	**Metric Name**	**%**
4.1	Percentage of the number of issues linked with other issues	24.67
4.2	Percentage of the number of issues (not pull request) linked with pull requests	6.74
4.3	Percentage of the number of pull requests linked with issues (not pull request)	20.55

Summary: Developers link 24.67% of issues with other issues (including pull requests). Following the natural flow from problem descriptions to solutions, we found that only 6.74% of the issues are linked with pull requests, but once we changed the direction (from the solution to the problem description), we retrieved 20.55% of links. The general average (Eq. 4.1) is greater than the other two equations, it means that developers link problem descriptions with not only related solutions, but also with related problems and solutions. For instance, to show that an issue is duplicated.

5.3 Answer of RQ2

RQ2: Do developers link commit with issues?

In Equation 4.4, we calculated the percentage of commits linked with issues, as represented in Figure 5.2a. There is a total of 934 070 commits and, 787 106 of them are linked with issues. It represents that 84.26% of the commits are linked with issues, as shown in Table 5.3. In Equation 4.5, we computed the percentage of commits linked with issues that are not pull requests, as represented in Figure 5.2b. The total number of commits linked with issues that are not pull requests is 108 553. It means that 11.62% of the commits are linked with issues that are not pull requests. In Equation 4.6, we explored the relation among commits and pull requests, as represented in Figure 5.2c. We found that 755 933 commits are linked with pull requests. It represents that 80.92% of the commits are linked with pull requests, as presented in Table 5.3. On GitHub the linking of commits with pull requests is automatic, since GitHub retrieve commits that belong to a branch that represents the pull request.



(a) Relation between commits and issues in both direction

(b) Relation between commits and issues (not PR) in both direction

(c) Relation between commits and pull requests in both direction

Figure 5.2: Relation between commits with pull requests and issues

In Equation 4.7, we calculated the percentage of links between issues and commits, as represented in Figure 5.2a. From a total of 693 292 issues, 350 733 of them are linked with commits. It represents that 50.58% of the issues are linked with commits. In Equation 4.8, we calculated the percentage of issues that are not pull requests linked with commits, as represented in Figure 5.2b. There is a total of 408 419 issues that are not pull requests, but only 96 375 of them are linked with commits. It represents that 23.59% of the issues (not pull requests) are linked with commits, as presented in Table 5.3. Practitioners should know that only one-quarter of the issues (not pull requests) are linked with commits. We recommend that they always refer the issues they are solving to the commits they created.

In Equation 4.9, we calculated the percentage of pull requests linked with commits, as represented in Figure 5.2c. There is a total of 284 873 pull request and 254 358 of them are linked with commits. It represents that 89.28% of the pull requests are linked commits, as presented in Table 5.3. We expect that developers solve their problems locally before creating pull requests, but our data shows that they normally create a pull request and start referring to the pull request in the commit message. Once they want to get feedback from others, this is a more natural working flow. Since others will see their code only when they pull to the main repository. It should explain this large of a percentage of pull requests linked with commits.

Equations 4.4 and 4.7 are important to answer this research question. The percentage of the number of links between commits and issues is greater than the percentage of the number of links between issues and commits. The reasons are: (i) when developers have a problem description (such as asking question or feedback), they link commits in the description of the issue, however in the other direction, they do not explicitly refer issues they are solving with commit messages, and (ii) in general the total number of commits is more than the total number of issues, therefore we get great number of commits linked with issues.

Table 5.3: Measures for metrics of research question 2

Equation	**Metric Name**	**%**
4.4	Percentage of the number of commits linked with issues	84.26
4.5	Percentage of the number of commits linked with issues (not pull requests)	11.62
4.6	Percentage of the number of commits linked with pull requests only	80.92
4.7	Percentage of the number of issues linked with commits	50.58
4.8	Percentage of the number of issues (not pull requests) linked with commits	23.59
4.9	Percentage of the number of pull requests linked with commits	89.28

Summary: Developers link 84.26% of the commits with issues, but in the other direction, they link 50.58% of the issues with commits. We found that 80.92% of the commits are linked with pull requests. On the other direction, we found that 89.28% of the pull requests in the commit messages. We believe that developers create pull requests and start referring them in the commit message. We found that 23.59% of the issues (not pull requests) are linked with commits whereas only 11.62% of the commits are linked with issues (not pull requests).

5.4 Answer of RQ3

RQ3: Do developers effectively use labels to represent the type of issues?
In Equation 4.10, we computed the percentage of issues linked with labels, as represented in Figure 5.3a. There is a total of 693 292 issues, and 412 092 of them are linked with labels. It represents that 59.43% of the issues are linked with labels. We recommend that practitioners link issues with labels to better organize tasks done in their projects.

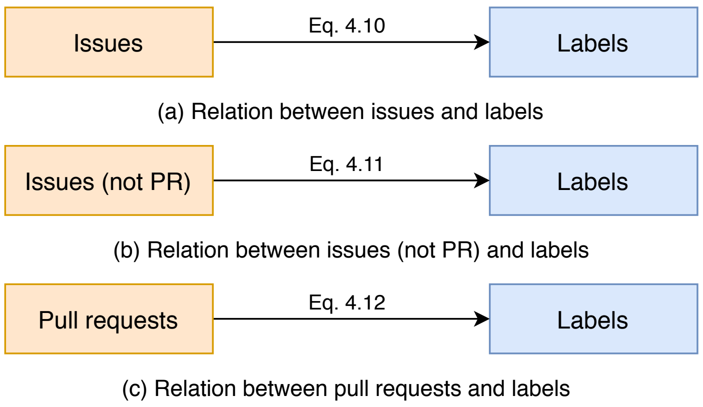

Issues
Eq. 4.10
Labels
(a) Relation between issues and labels
Issues (not PR)
Eq. 4.11
Labels
(b) Relation between issues (not PR) and labels
Pull requests
Eq. 4.12
Labels
(c) Relation between pull requests and labels

Figure 5.3: Relation between labels with pull requests and issues

In Equation 4.11, we obtained the percentage of issues that are not pull requests linked with labels represented in Figure 5.3b. There is a total of 408 419 issues that are not pull requests and 236 592 of them are linked with labels. It represents that 57.92% of the issues (not pull requests) are linked with labels, as presented in Table 5.4. In Equation 4.12, we calculated the percentage of pull requests linked with labels represented in Figure 5.3c. There is a total of 284 873 pull requests and 175 500 of them are linked with labels. It represents that 61.60% of the pull requests are linked with labels, as presented in Table 5.4. As pull requests represents code changes that developers did to solve an issue, we recommend that developers link pull requests with labels because these links will be useful for them to find similar solutions in the future.

Table 5.4: Measures for metrics of research question 3

Equation	**Metric Name**	**%**
4.10	Percentage of the number of issues linked with labels	59.43
4.11	Percentage of the number of issues (not pull request) linked with labels	57.92
4.12	Percentage of the number of pull requests linked with labels	61.60

Summary: Developers link 59.43% of the issues with labels (Eq. 4.10). They also refer labels to 57.92% of the issues that are not pull requests (Eq. 4.11). Moreover, developers link 61.60% of the pull requests with labels (Eq. 4.12). We recommend that practitioners link more labels with social assets to make easier the filtering process of issues and pull requests, and support the evolution of the project.

6 Discussion

In this chapter, we present a comparison with previous studies (Section 6.1). In Section 6.2, we compare top labels used in general, in issues (not pull requests), and in pull requests. In Section 6.3, we present a survey to investigate developers' opinion about linking social and technical assets on GitHub. In Section 6.4, we present internal and external threats to validity our study.

6.1 Comparison with Previous Studies

In this section, we compare our findings on GitHub coordination with related work. Zhang et al. [Zha+18] explored links between issues and other related issues within the project or across projects. They describe that linking issues may enhance discussion, avoid duplication, and even hasten the resolution of the original issue. They determined that developers link 22.4% of the issues with related issues in a project. In our study, we also explored the relationship between issues and related issues. We found that 24.67% of the problem are linked with other similar problems. Different from them, we explored other relationships such as issues (not pull requests) with pull requests, pull requests with issues (not pull requests), commits with issues, and issues with labels. Furthermore, we conducted a survey to get developers' opinion about their use of issues, pull requests, labels as well as how they relate these assets.

In another study, Tien-Duy et al. [Le+15] proposed an approach to predict links among issues and commits. Since, the links between issue reports and their corresponding commits are often missing. These links are important for software maintenance tasks including assessing the reliability of a particular part of a software system or predicting future defective software components. They presented a novel bug linking technique and compared their approach to MLink, which is the latest state-of-the-art bug linking approach. They found that 89% of the links were predicted correctly. We explored the relationship among issues and commits, and found that 84.26% of the commits are linked with issues. Moreover, we explored other relationships such as commits with pull requests, and commits with issues (not pull requests) in both directions. We also conducted a survey to get developers' opinion about these relationships.

6.2 Comparing Top 10 Labels Used in General, in Issues (Not Pull Requests), and in Pull Requests

In Table 6.1, we present the list of top 10 labels used in general, in issues that are not pull requests, and in pull requests. We see that top 3 labels most used in general are: *bug*, *need_triage*, and *cla: yes*. The top 3 labels most used in issues (not pull requests) are: *bug*, *FrozenDueToAge*, and *question*. And, the top 3 labels most used in pull requests are: *cla: yes*, *need_triage*, and *bug*.

From the top 10 labels in general, we found 4 labels that are often assigned with both issues (not pull requests) and pull requests. These labels are: *bug*, *needs_triage*, *module*, and *support: core*. A bug report normally requires code changes and when developers are coding they will refer it to the pull request. We found 6 labels that are normally used in issues (not pull requests). These labels are: *FrozenDueToAge, question, fixed, resolved, duplicate*, and *stale*. To illustrate, *question* label is used 14 859 times and 99.67% of the references are with issues that are not pull requests. Likewise, *FrozenDueToAge* label is used 24 028 times and 99.69% of the references are with issues (not pull requests). We assume that these labels contain problem description that do not need coding as questions when using or setting up the project.

Table 6.1: List of top 10 labels used in general, in issues (not PR), and in PR

Top	**Label Name**	**#general**	**Label Name**	**#Issues (not PR)**	**Label Name**	**#PR**
1	bug	52 534	bug	35 488	cla: yes	32 304
2	needs_triage	40 825	FrozenDue ToAge	23 955	needs_triage	26 514
3	cla: yes	32 309	question	14 810	bug	17 046
4	module	27 854	needs_triage	14 311	module	16 993
5	FrozenDue ToAge	24 028	fixed	11 442	support: community	15 293
6	support:core	23 436	module	10 861	CLA Signed	14 134
7	support: community	21 458	Resolved	10 457	support:core	13 792
8	question	14 859	duplicate	10 419	needs_ revision	13 644
9	CLA Signed	14 137	support:core	9 644	community_ review	11 057
10	needs_ revision	13 648	stale	9 512	Test	10 804

# stands for the number of times the label was used in, PR stands for pull requests

We found 6 labels that are normally used in pull requests. These labels are: *cla: yes, support: community, CLA Signed, needs_revision, community_review*, and *test*. To illustrate, *cla: yes* labels is used 32 309 time and 99.98% of the references are with pull requests. Similarly, *CLA Signed* labels is used 14 137 and 99.97% of the references are with pull requests. A contribution license agreement (CLA) is a legal document which allows developers to contribute to a project. Developers must sign a CLA before a merging a pull request. Therefore, CLA signed is often used in pull requests. It is interesting to find labels that are more related with code because researchers may use this information to refine communication more related to code changes and use it in their analysis.

6.3 A Survey to Investigate Developers' Opinion about Linking Social and Technical Assets on GitHub

Based on the results from our empirical study, we conducted a survey to understand developers' opinion about linking social and technical assets on GitHub. In Section 6.3.1, we describe how we design and plan the survey. In Section 6.3.2, we present the results of the survey. In Section 6.3.3, we compare the results of our empirical study with the results of the survey.

6.3.1 Design and Planning

We defined 26 questions on Google form and categorized them into 5 groups: (i) Participant's background, to know the background knowledge of developers (e.g., work experience, working industry, role in industry, education, and gender), (ii) Participants'

GitHub knowledge, to know developers' experience and knowledge of GitHub features (i.e., how often they use, purpose to use, contribution with others, size of team, and familiarity with issues, pull requests, commits, labels), (iii) Participants' report of use of issues and pull requests, to know whether and why developers link issues with pull requests, (iv) Participants' report of use of commits, to know whether and why developers link commits with issues, and (v) Participants' report of use of labels, to know whether and why developers assign labels to pull requests and issues. The complete survey can be found in Appendix A.

We send 20 emails to the developers (students and colleagues) who use GitHub frequently and ask them to participate. We also made the survey available on seven web pages and groups on social platforms (Facebook and Twitter) where GitHub developers are active. These groups and platforms include GitHub Education[1], GitHub for Beginners[2], GitHub Social Coding[3], GitHub Community[4], GitHub Status[5], GitHub Trends[6], and GitHub API[7]. The survey was online for 15 days from February 10th to February 25th of 2020.

6.3.2 Data Collection and Analysis

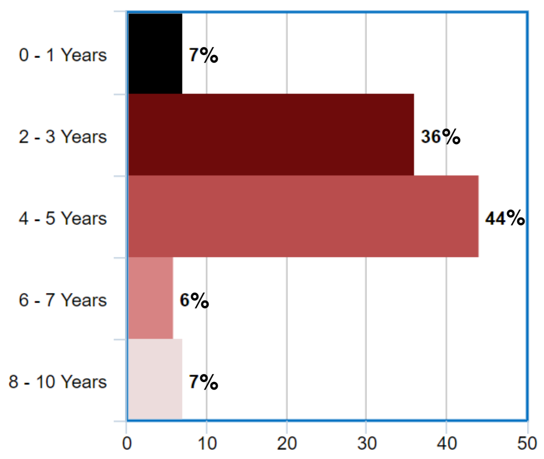
We received responses from 109 participants of which we excluded 1 response because the participant does not agree to participate in the research study, and we excluded 8 responses because the participants did not have experience with GitHub. At the end,

[1]https://www.facebook.com/GitHubEducation/
[2]https://www.facebook.com/groups/githubforbeginners/
[3]https://www.facebook.com/groups/githubsocialcoding/
[4]https://twitter.com/GitHubCommunity
[5]https://twitter.com/githubstatus
[6]https://www.facebook.com/github.trends/
[7]https://twitter.com/GitHubAPI

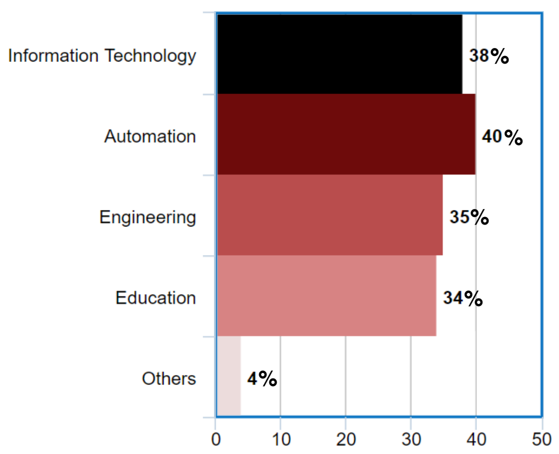
we use data from 100 participants. Next, we present an overview of our data based on the 5 groups of our survey (see Section 6.3.1).

Participants' general background: In Figure 6.1a, we show that 7% of the participants have from 0 to 1 years of experience, 36% of the participants have from 2 to 3 years of experience, 44% of the participants have from 4 to 5 years of experience, 6% of the participants have from 6 to 7 years of experience, and 7% of the participants have from 8 to 10 years of experience . In Figure 6.1b, we see that most of the participants have worked in information technology (38%), automation (40%), engineering (35%), education (34%), and other industries (4%). In Figure 6.1c, we show that 37.8% of the participants work as a software developer, 34.7% of participant work as a web developer, 42.9% of the participants work as a software tester, 38.8% of participant work as a android developer, and 9% of participant work as a others. In Figure 6.1d, we illustrate that 65% of the participants have advanced degree such as Master, Ph.D., M.D., 27% of the participants have bachelor degree, 4% of the participants have associate degree, 3% of the participants have Some college, no degree, and 1% of the participants have graduate high school. Hence, we see that most of the participants are highly qualified. In Figure 6.1e, we show that 55% of the participants are males and 45% of the participants are females. Despite males are the majority also in real world, we have a great number of females participating of our survey.

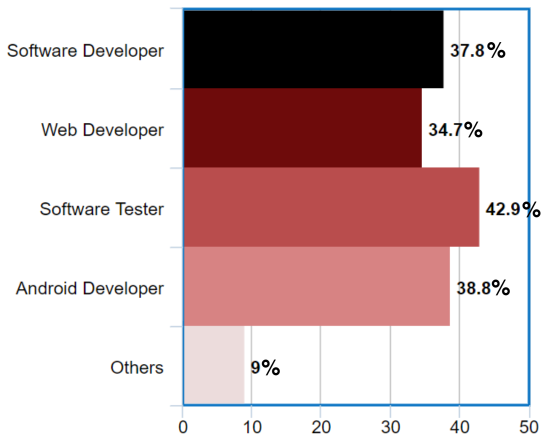
Participants' GitHub knowledge: In Figure 6.2a, we show that 47.5% of the participants always use GitHub, 28.3% of the participants normally use GitHub, and 24.2% of the participants sometimes use GitHub. In Figure 6.2b, we show that most of the participants use GitHub for organizational work (59.2%), educational work (53.1%), personal work (58.2%), and others (1%). In Figure 6.2c, we show that 78% of the participants contribute with others on GitHub and 22% of the participant do not contribute



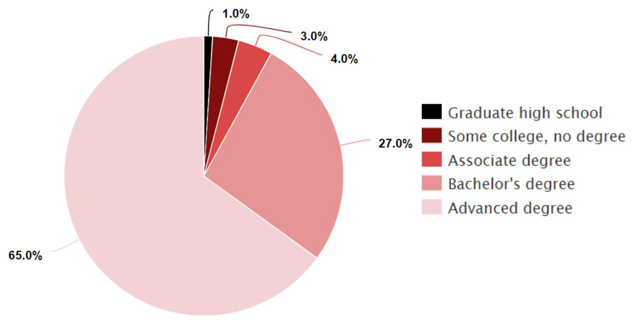
(a) Percentage of developers' experience



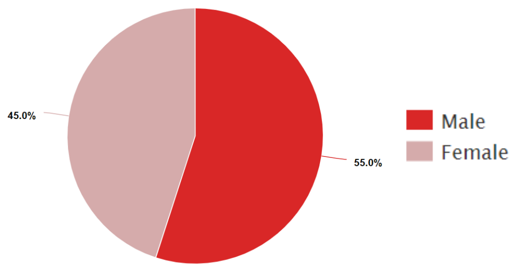
(b) Percentage of developers in industry



(c) Percentage of developers' role in Industry



(d) Educational background



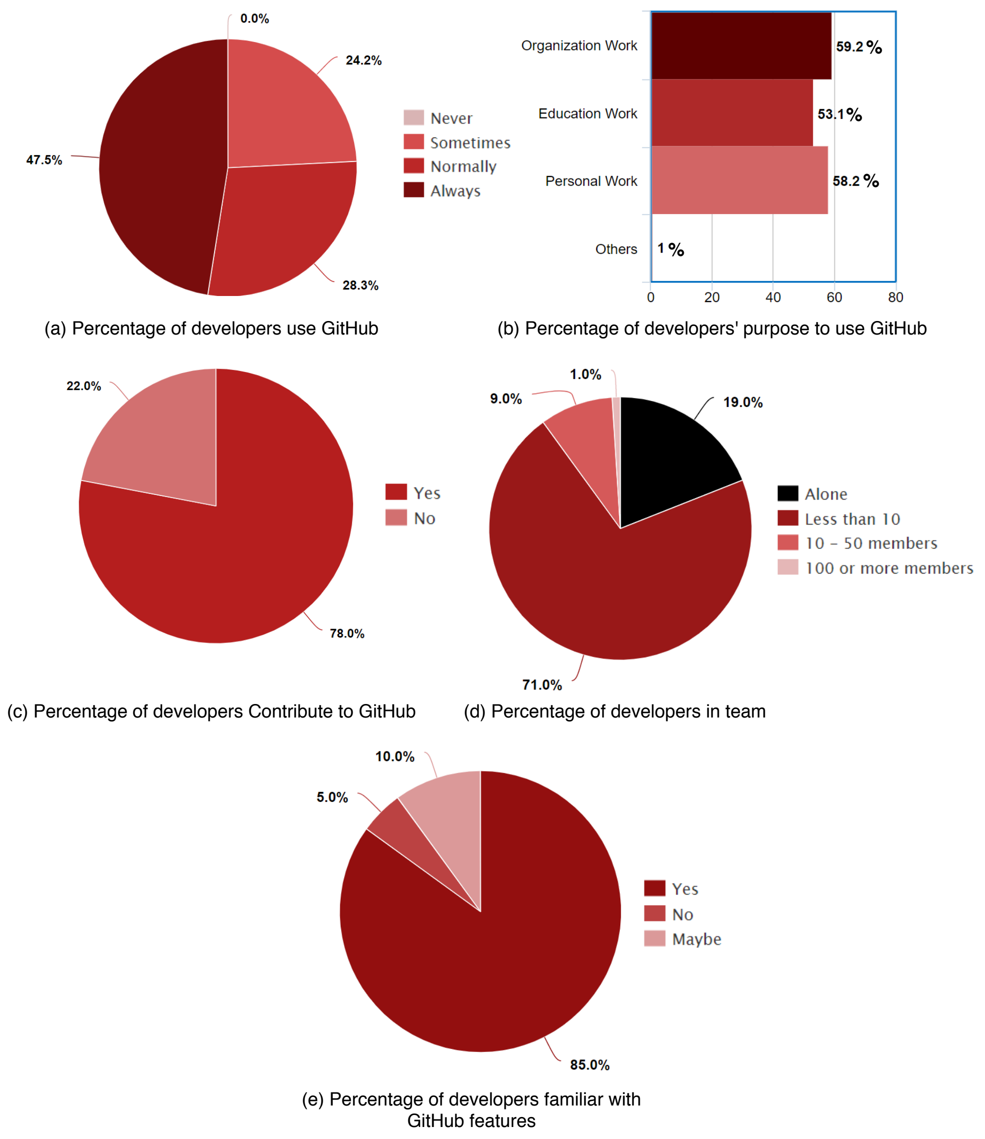
(e) Percentage of gender

Figure 6.1: Results of participants' general background

with others. In Figure 6.2d, we show that 19% of the participants contribute alone on GitHub, 71% of the participants contribute in teams having less than 10 members, 9% of the participants contribute in teams having members from 10 to 50, and only 1% of the participants contribute in teams having 100 or more members. In Figure 6.2e, we show that 85% of the participants are familiar with GitHub issues, pull requests, commits, and labels, 10% of them do not feel confident about their knowledge on these features, and 5% of them are not familiar with all of these features.

Participants' report of use of issues and pull requests: In Figure 6.3a, we show that 52% of the participants create issues yearly, 18% of the participants create issues monthly, 11% of the participants create issues weekly, 3% of the participants create issues daily, and 16% of the participants never create issues. In Figure 6.3b, we show that 85% of the participants created less than 5 issues last month, 10% of the participants created from 5 to 20 issues last month, 2% of the participants created between 20 and 50 issues last month, and 3% of the participants created more than 50 issues last month. In Figure 6.3c, we show that 51% of the participants creates pull requests yearly, 19% of the participants create pull requests monthly, 10% of the participants create pull requests weekly, 4% of the participants create pull requests daily, and 16% of the participants do not create pull requests. In Figure 6.3d, we show that 83.7% of the participants created less than 5 pull requests last month, 11.2% of the participants created from 5 to 20 pull requests last month, 3% of the participants created between 20 and 50 pull requests last month, and only 2% of the participants created more than 50 pull requests last month.

In Figure 6.3e, we show that 48% of the participants sometimes refer issues with pull requests, 5% of the participants normally refer issues with pull requests, 5% of the participants always refer issues with pull requests, and 42% of the participants do not refer issues with pull requests. In Figure 6.3f, we show that 41% of the participants



(a) Percentage of developers use GitHub

(b) Percentage of developers' purpose to use GitHub

(c) Percentage of developers Contribute to GitHub

(d) Percentage of developers in team

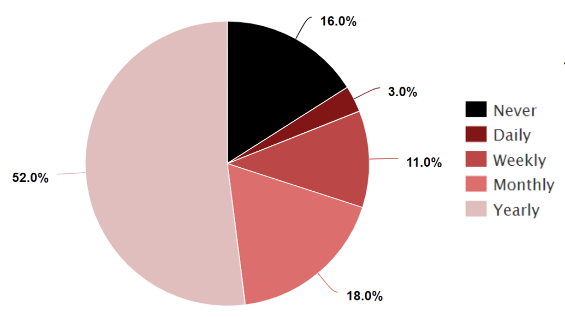
(e) Percentage of developers familiar with GitHub features

Figure 6.2: Results of participants' GitHub knowledge

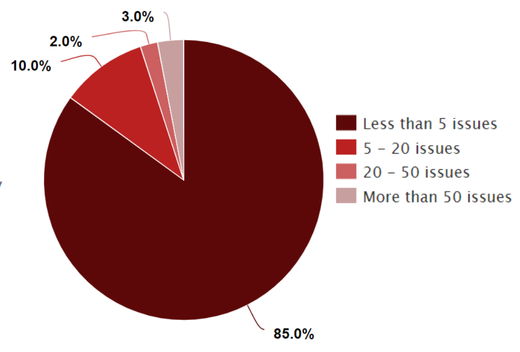
sometimes refer pull requests with issues, 13% of the participants normally refer pull requests with issues, 3% of the participants always refer pull requests with issues, and 43% of the participants do not refer pull requests with issues. In Figure 6.3g, we show that 43% of the participants sometimes refer issues with related issues, 10% of the participants normally refer issues with related issues, 6% of the participants always refer issue with related issues, and 41% of the participants do not refer issues with related issues. In Figure 6.3h, we show that more than 82% of the participants who believe that relating pull requests with issues is useful for project coordination, 16% of the participants who do not feel confident that relating pull requests with issues is useful for project coordination, and only 2% of the participants who do not think that relating pull requests with issues is useful for project coordination. For instance, P48 said that relating issues and pull requests help to keep track of the changes of a project. P41 and P44 said that these links are useful to find problems and related solutions.

Participants' report of use of commits: In Figure 6.4a, we show that 41.4% of the participants daily create commits, 31.3% of the participants weekly create commits, 12.1% of the participants monthly create commits, 10.1% of the participants yearly create commits, and 5.1% of the participants do not create commits. In Figure 6.4b, we show that 75.8% of the participants created less 10 commits last month, 20.2% of the participants created from 10 to 20 commits last month, 3% of the participants created between 20 to 50 commits last month, and only 1% of the participants created more than 100 commits last month.

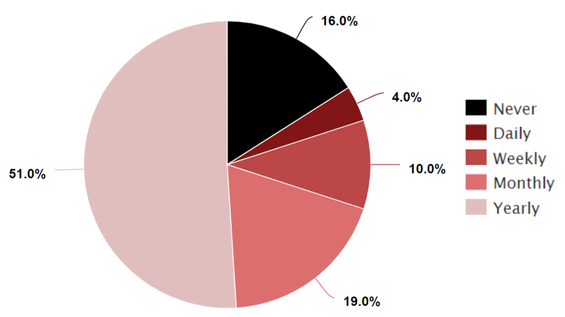
In Figure 6.4c, we show that 54% of the participants always refer issues with commits, 17% of the participants normally refer issues with commits, 23% of the participants sometimes refer issues with commits, and 6% of the participants do not refer issues with commits. In Figure 6.4d, we see that 48% of the participants always refer commits in the



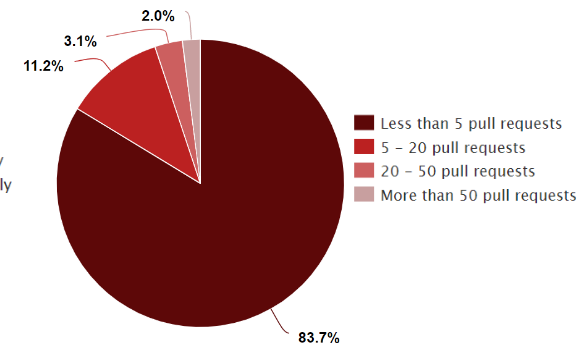
(a) Percentage of developers create issues



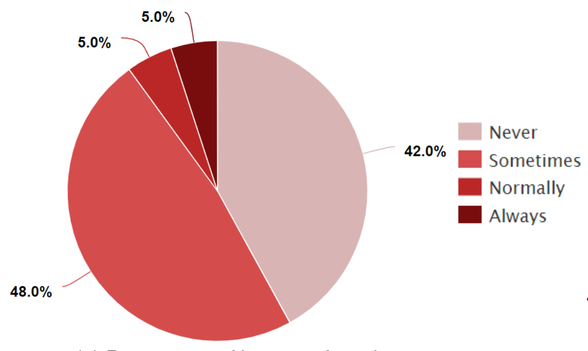
(b) Percentage of developers create issues last month



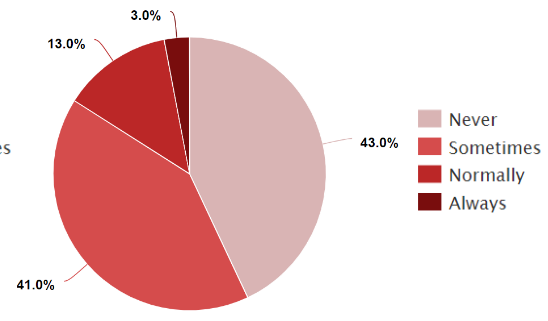
(c) Percentage of developers create pull requests



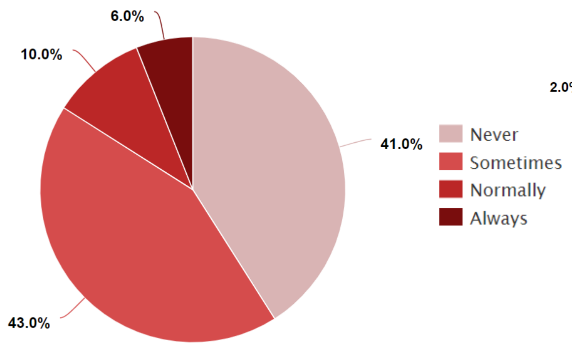
(d) Percentage of developers create pull requests last month



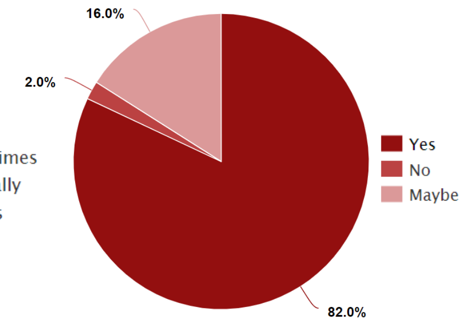
(e) Percentage of issues referred in pull requests



(f) Percentage of pull requests referred in issues



(g) Percentage of issues referred in other issues

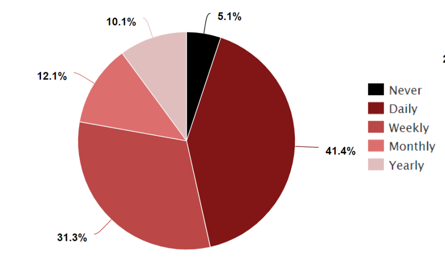


(h) Percentage of developers think links between pull requests and issues are useful

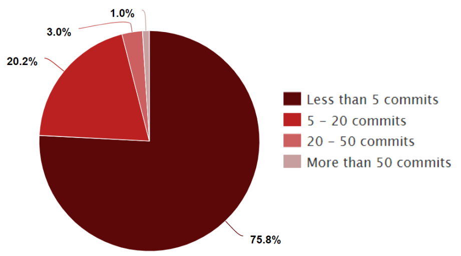
Figure 6.3: Results of participants' report of use of issues and pull requests

description of issues, 15% of the participants normally refer commits in the description of issues, 21% of the participants sometimes refer commits in the description of issues, and 16% of the participants do not refer commits in the description of issues. In Figure 6.4e, we show that more than 86% of the participants who think that relating commits to issues are useful for project coordination, 13% of the participants who do not feel confident that relating commits to issues are useful for project coordination, and only 1% of the participants who do not think that relating commits to issues are useful for project coordination. For instance, P9 said that links among commits and issues are important because it makes easier to retrieve code changes. P18 mentioned that referring issues with commits is helpful for code review and tracking solutions.

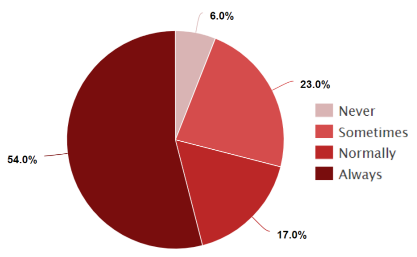
Participants' report of use of labels: In Figure 6.5a, we show that 62% and 67% of the participants use default and customized labels, respectively, and 13% of the participants do not use labels. In Figure 6.5b, we show that 63% of the participants assign labels to issues and to pull requests, and 22% of the participants do not assign labels. In Figure 6.5c, we show that 79% of the participants who think that assigning labels to issues or pull requests is useful for project coordination, 14% of the participants who do not feel confident that assigning labels to issues or pull requests is useful for project coordination, and only 7% of the participants who do not think that assigning labels to issues or pull requests is useful for project coordination. For instance, P9 and P15 said that assigning labels to issues make filtering easy and help in categorizing multiple problem descriptions. Two other (P22 and P25) mentioned that labels are useful to distinguish issues from one another and for better organize the work of the project.



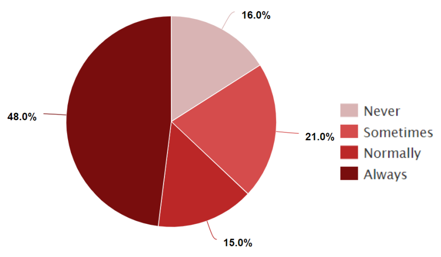
(a) Percentage of developers create commits



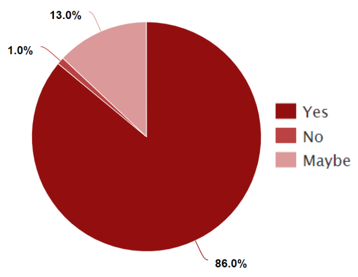
(b) Percentage of developers create commits last month



(c) Percentage of issues referred in commits

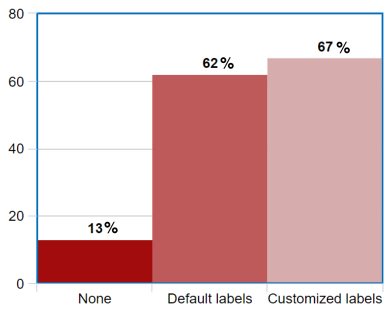


(d) Percentage of commits referred in issues

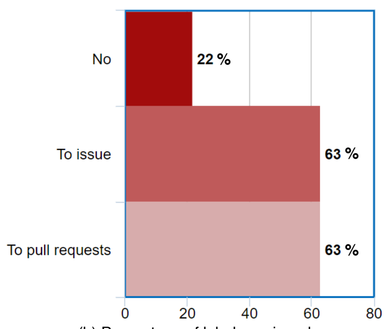


(e) Percentage of developers think links between commits and issues are useful

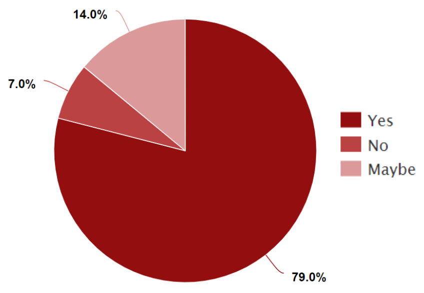
Figure 6.4: Results participants' report of use of commits



(a) Percentage of labels used by developers



(b) Percentage of labels assigned to GitHub assets



(c) Percentage of developers who think labels are useful

Figure 6.5: Results participants' report of use of labels

6.3.3 Evaluation

In this section, we compare the use of GitHub features by the participants of the survey and the results of our empirical study with the results of the survey.

Discussion: Looking at the results of our survey, we can see that around 80% of the participants agree that linking social and technical assets is important for the project coordination. However, the participants reported that 41.4% of them create commits daily and 31.3% of them create commits weekly. We found that 52% and 51% of the participants create issues and pull requests yearly, respectively. Regarding relationships among GitHub features, we see that 5% of participants always link issues with pull requests, 3% of the participants always link issues with pull requests, and 6% of participants always link issues with other issues in practice. On the other hand, 54% of the participants always link issues with commits and 48% of the participants always link commits with issues. This way, we conclude that participants are more familiar with commits and as expected issues and pull requests need more time to be addressed and are normally created by core developers of projects hosted on GitHub, because normally only them have permission to create these assets.

Research question 1: In Section 5.2, we showed that developers link 24.67% of the problems with related problems and solutions. In the survey, we found that 59% of the participants (43% sometimes + 10% normally + 6% always) refer issues with related issues. Our empirical study shows that developers link 20.55% of the pull request with issues, and the participants of the survey reported that 57% of them (41% sometimes + 13% normally + 3% always) link pull requests in the description of issues. Furthermore, in Section 5.2, we showed that 6.74% of the issues are linked with pull requests . In the survey, we found 58% of the participants (48% sometimes + 5% normally + 5% always) refer issues in the body of pull requests. Developers think that relating issues with pull requests provide contextual information, and allow tracking of problem and solutions. However, they do not link these assets often in practice.

Research question 2: Our empirical study shows that 84.26% of the developers link commits in the description of the issues. In the survey, we found that 84% of the participants (21% sometimes + 15% normally + 48% always) link commits with issues while addressing them. In other direction, we found in empirical study that developers link 50.58% of the issues with commits (see in Section 5.3). In the survey, we found that 94% of the participants (23% sometimes + 17% normally + 54% always) always refer issues in the commit message. Developers believe that linking commits with issues in both directions makes it easier to retrieve code changes and tracking partial solutions.

Research question 3: In Section 5.4, we found that developers link 59.43% of the issues with labels and 61.60% of the pull requests with labels. In the survey, we found that 63% of the participants relate issues and pull requests with labels. They reported that labeling issues and pull requests allow them to categorize, filter, and distinguish issues/pull requests.

The results from the survey are similar to what we found empirically. Around 80% of the participants of our survey agreed that links among issues, pull requests, commits, and labels are useful. However, despite of the importance, they do not relate social and technical assets very often in practice.

6.4 Threats to Validity

In this section, we discuss internal and external threats to validity our study. We eliminate threats when possible and decrease their effect when the elimination was not possible.

6.4.1 Internal Validity

Internal validity is threatened by five factors. First, we selected projects from different programming languages and one language could have dominated our dataset. To minimize this threat, we check if one programming language dominated our dataset and excluded the less popular *JavaScript* projects, as described in Section 4.2. Second, we looked at only one social network platform. However, other platforms such as GitLab, Bitbucket the links between social and technical assets may be different. We choose GitHub because it is one of the most used social coding platform, hence we believe that this platform reflects the real working practice. Anyway, our results are valid only for GitHub projects.

Third, we get data from all developers, however, some developers may have more experience than others and it may also impact our analysis. Since we want to provide a general overview, the experience of developers do not affect our analysis. Fourth, we do not check the correctness of retrieved links. We believe that our links are correct because we retrieve links that are explicitly presented in the GitHub. Nevertheless, if developers created false links we retrieved them. Fifth, we only look at links between issues, pull requests, commits, and labels from the same projects. We do not retrieve links across projects. Looking at links across projects is out of our scope, but more links can be found and support developers addressing similar tasks.

6.4.2 External Validity

External validity is threatened by three factors. First, we explored only GitHub as a platform to understand how developers link technical and social assets. Generalization of our study with other platforms, projects, and development models is limited. This limitation of the sample was necessary to reduce the influence of confounds, increasing internal validity, though [SS15]. More research is required to generalize with other version control systems, development models, and platforms. We believe that we selected and analyzed a practically relevant platform and a substantial number of software projects from various domains, and coordination practices. In addition, our filters applied during subject project selection (Section 4.2) guarantee, that we sampled projects that actively use GitHub as a social coding tool, for instance.

Second, we selected 66 GitHub repositories and collected data as the number of issues, pull requests, commits, and labels. However, these numbers change daily with the progress of the repository. Hence, our results are valid for the period of our analysis (from the project creation up to November 2019). Anyway, we believe that our results remain true for other time slots.

Third, we conducted a survey based on our empirical study and received responses from developers. We assume that participants have knowledge of GitHub features. To mitigate this threat, we filtered participants based on their experience with GitHub. We excluded participants who have no experience with GitHub and who do not consent to participate in the survey.

7 Final Remarks

7.1 Conclusion

Open source software development is often characterized as a fundamentally new way to develop software [Fri+02]. Software development is a collaborative activity where success depends on the ability to coordinate social and technical assets. GitHub is one of the most used social coding platform by developers who contribute to collaborative projects in an open manner [Sto+17]. Several studies investigate the developers' behavior on GitHub. However, none of them has investigated which GitHub features developers use to support their coordination and how they link social and technical assets in practice. We defined three research questions (see Section 4.1) based on how developers relate assets on GitHub. There are several features on GitHub like issues, pull requests, commits, and labels, which facilitate coordination among developers. We select 66 GitHub repositories and collect data from them, as described in Section 4.2. We investigated three relationships among GitHub features: (i) whether developers link their problem descriptions (issues) with the problems they solved (pull requests), (ii) whether developers link commits with the problems their described (issues) and addressed (pull requests), and (iii) whether developers assign labels to problems (issues) and solutions (pull requests).

Furthermore, we use 16 variables and formulated 12 equations to support our answers to the three research questions, as described in Section 4.4.

To answer research question 1, we investigated links between issues and pull requests. We found that developers link 24.67% of the problems with related problems and solutions. Following the natural flow from problems to the solutions, we found that only 6.74% of the issues are linked with pull requests. Once we changed the direction (from solution to problem description) we retrieved 20.55% links. To answer research question 2, we investigated links among commits with issues and pull requests. We found that developers link 84.26% of the commits in the description of the issues, but in the other direction they link 50.58% of the issues with commits. Developers link 80.92% of the commits in pull requests. On the other direction, we found that 89.28% of the pull requests in the commit messages. Moreover, we found that 23.59% of the issues (not pull requests) are linked with commits whereas only 11.62% of the commits are linked with issues that are not pull requests. To answer research question 3, we investigate links between labels with issues and pull requests. We found that 59.43% of the developers link issues with labels. They also refer labels with 57.92% of the issues that are not pull requests. Furthermore, 61.60% of the developers link pull requests with labels.

After our empirical study, we conducted a survey and to investigate if GitHub users agree with our findings. We got 100 valid responses. The survey contains 5 groups to explore participants' background and their experience with GitHub features. Regarding links between issues and pull requests, we found that: (i) 59% of the participants relate problems with related problems, (ii) 58% of the participants refer issues in the body of pull requests, and (iii) 57% of the participants describe pull requests in the description of issues. Regarding links among issues and commits, we found that: (i) 84% of the participants refer commits in the description of issues, and (ii) 94% of the participants refer problems in the commit messages. Regarding links among issues and labels, we found that participants use both default and customized labels, and in 63% of the cases they assign labels to issues and pull requests.

We realized that developers link 20.55% of the problem they addressed (issues) with the solution they proposed (pull requests). However, we expect more links between issues and pull requests. Hence, we encourage that developers should specify in the description of issues which pull requests solve the problem. For instance, after finishing an issue they should come back to the issue and comment the number of the pull request that fix the target issue. In the other direction, they should also describe in the body of pull requests which issues are addressed. Moreover, developers should explicitly link issues with related issues. Developers link 84.26% of commits with issues whereas in the other direction they link 50.58% of the issues with commits. We recommend developers that they should link issues with commits by writing issues id in commit messages. They should also link commits with issues by writing commit hash in the description of issues. Furthermore, we realized that developers link labels with 59.43% of the issues. We encourage developers to assign labels to more issues to better organize project work and make easier the filtering process of issues. In general, developers agree that creating links between social and technical assets helps in project coordination, however, they do not relate assets very often in practice. Therefore, we suggest that developers should create more links among social and technical assets to increase coordination.

7.2 Contributions

This thesis makes the following main contributions:

- We empirically present evidence that one quarter (24.67%) of the developers link problem descriptions with solutions.

- We provide evidence that 84.26% of the developers refer commits in the description of issues.

- We found that developers link 59.43% of the problem descriptions (i.e., issues) with labels and 61.60% of the pull requests with labels.

- We show that 80% of the participants of our survey agreed that linking social and technical assets is important for project coordination, however they do not link assets very often in practice.

- We provide a list of the top 10 labels used in general, in issues (not pull requests), and in pull requests. As result, we found that: *cla: yes, CLA Signed* labels are the two most related to code (pull requests). *FrozenDueToAge, question* labels are the two most related to communication (issues). And, *bug, needs_traige, module* labels are related to both issues and pull requests. Assigning labels to assets help developers to refine communication related to code changes.

- We make our infrastructure publicly available, and the data is also publicly available [Ver20].

7.3 Future Work

As suggestions of future work, we see four promising directions. First, we suggest to replicate our study in other social network platforms such as GitLab and Bitbucket to determine developers' coordination and how it impacts in our analysis. Second, we suggest to divide our projects in the different programming languages to see if developers link differently depending on the programming language. Third, we suggest a study investigating the use of labels in issues and pull requests to find which labels are more and less related to code. This kind of study could help researchers to refine communication related to code changes. Fourth, we suggest extending the survey to investigate additional techno-social challenges faced by developers. Asking developers personally may help us to understand their reasons to ignore linking social and technical assets.

A Survey

In this appendix, we present the survey that investigate developers' opinion about linking social and technical assets. The results of the survey were presented in Section 6.3. It was divided into 6 sections: (i) Survey on GitHub communication, (ii) Background, (iii) GitHub knowledge, (iv) GitHub issues and pull requests, (v) GitHub commits, and (vi) GitHub labels.

Section 1: Survey on GitHub Communication

You are receiving this survey as our initial investigation identified you as a GitHub user. In this survey, we want to know your opinion regarding importance and adaptability of GitHub features for communication among developers. We plan to include the results of this survey in a scientific publication. Should you be interested in being informed about the outcome of this study or any resulting publication, you will be provided an opportunity to indicate this and provide us with your email address.

Electronic consent: Please select your choice below. Selecting the "yes" option below indicates that: i) you have read and understood the above information, ii) you voluntarily agree to participate, iii) you agree that your answers can be used for research purpose, and iv) you are at least 18 years old. If you do not wish to participate in the research study, please decline participation by selecting "No". I consent to participate in this research study.

( ) Yes

( ) No

Section 2: Background

1. How many years experience you have in software development?

   (ex: 0, 1, 2, 3...10...)

2. In which industry you are working / have worked?

   ( ) Information Technology

   ( ) Automation

   ( ) Engineering

   ( ) Education

   ( ) Other

3. What of these roles fits with you in industry?

   ( ) Software Developer

   ( ) Web Developer

   ( ) Software Tester

   ( ) Android Developer

   ( ) Other

4. What is your level of education?

   ( ) Less than high school

   ( ) Graduate high school

   ( ) Some college, no degree

   ( ) Associate degree

   ( ) Bachelor's degree

( ) Advanced degree (Master's, Ph.D., M.D.)

( ) Other

5. What is your gender?

   ( ) Male

   ( ) Female

   ( ) Prefer not to say

Section 3: GitHub Knowledge

6. How often do you use GitHub?

   ( ) Never

   ( ) Sometimes

   ( ) Normally

   ( ) Always

7. What is your purpose of use of GitHub?

   ( ) Organizational work

   ( ) Educational work

   ( ) Personal work

   ( ) Other

8. Do you normally contribute with others in GitHub?

   ( ) Yes

   ( ) No

9. With how many members compose your team?

   ( ) Alone

   ( ) Less than 10

( ) 10-50 members

( ) 100 or more members

10. Are you familiar with GitHub features such as issues, pull requests, commits and labels?

    ( ) Yes

    ( ) No

    ( ) Maybe

Section 4: GitHub Issues and Pull Requests

11. How often do you create issues?

    ( ) Never

    ( ) Daily

    ( ) Weekly

    ( ) Monthly

    ( ) Yearly

12. How many issues did you create on GitHub in last month?

    ( ) Less than 5 issues

    ( ) 5-20 issues

    ( ) 20-50 issues

    ( ) More than 50 issues

13. How often do you create pull requests?

    ( ) Never

    ( ) Daily

    ( ) Weekly

( ) Monthly

( ) Yearly

14. How many pull requests did you create on GitHub in last month?

    ( ) Less than 5 pull requests

    ( ) 5-20 pull requests

    ( ) 20-50 pull requests

    ( ) More than 50 pull requests

15. While solving an issue, do you normally refer to the issue in the description or body of the pull request you are creating? Why do you think it is important?

    ( ) Never

    ( ) Sometimes

    ( ) Normally

    ( ) Always

    Comment: ______________________

16. Before close an issues that is solved, do you explicitly describe which pull requests solve the problem? Why do you think it is important?

    ( ) Never

    ( ) Sometimes

    ( ) Normally

    ( ) Always

    Comment: ______________________

17. Do you describe in the body of an issue about similar or related issues? Why do you think it is important?

    ( ) Never

( ) Sometimes

( ) Normally

( ) Always

Comment: ________________________

18. Do you think that relating pull requests with issues is useful for project coordination?

    ( ) Yes

    ( ) No

    ( ) Maybe

    Comment: ________________________

Section 5: GitHub Commits

19. How often do you commit on GitHub?

    ( ) Never

    ( ) Daily

    ( ) Weekly

    ( ) Monthly

    ( ) Yearly

20. How many commits did you do on GitHub in last month?

    ( ) Less than 10 commits

    ( ) 10-20 commits

    ( ) 20-50 commits

    ( ) More than 50 commits

21. Do you explicitly refer to the issue you are addressing in the commit message?

    ( ) Never

( ) Sometimes

( ) Normally

( ) Always

Comment: ______________________

22. In the issue body or description, do you explicitly refer the commit hash related to it?

    ( ) Never

    ( ) Sometimes

    ( ) Normally

    ( ) Always

    Comment: ______________________

23. Do you think that relating commits to issues are useful for the project coordination?

    ( ) Yes

    ( ) No

    ( ) Maybe

    Comment: ______________________

Section 6: GitHub Labels

24. Which type of labels do you use?

    ( ) Default

    ( ) Customized

    ( ) None

25. Do you assign labels on GitHub?

    ( ) No

( ) To issues

( ) To pull requests

26. 26. Do you think that assigning labels to issues or pull requests are useful for the project coordination?

    ( ) Yes

    ( ) No

    ( ) Maybe

    Comment: ______________________

Section 7: Email Information (optional)

Please enter your e-mail if you would like to receive a copy of our study. We will not link your e-mail address to your survey responses, nor will we publish your e-mail address in any which way. Participation in the survey is voluntary. If you do not wish to participate, please go ahead and submit the survey below. If you do wish to participate, please enter your e-mail address and then submit the survey. We thank you for your participation in this research!

____

Bibliography

[AB02] Ulf Asklund and Lars Bendix. “A study of configuration management in open source software projects”. In: *IEE Proceedings-Software* 149.1 (Feb. 2002), pp. 40–46. URL: `https://digital-library.theiet.org/content/journals/10.1049/ip-sen_20020196` (cit. on p. 10).

[Bir+08] Christian Bird et al. “Latent Social Structure in Open Source Projects”. In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. SIGSOFT ’08/FSE-16. Atlanta, Georgia: Association for Computing Machinery, Nov. 2008, pp. 24–35. ISBN: 9781595939951. DOI: `10.1145/1453101.1453107`. URL: `https://doi.org/10.1145/1453101.1453107` (cit. on pp. 21, 24).

[Bir+09] Christian Bird et al. “Does distributed development affect software quality? An empirical case study of Windows Vista”. In: *Proceedings of the 31st international conference on software engineering*. IEEE Computer Society. June 2009, pp. 518–528. ISBN: 978-1-4244-3453-4. DOI: `10.1109/ICSE.2009.5070550` (cit. on p. 1).

[Bis+13] Tegawendé F Bissyandé et al. "Got issues? who cares about it? a large scale investigation of issue trackers from github". In: *2013 IEEE 24th international symposium on software reliability engineering (ISSRE).* IEEE. Jan. 2013, pp. 188–197. ISBN: 978-1-4799-2366-3. DOI: `10.1109/ISSRE.2013.6698918` (cit. on p. 16).

[BV18] Hudson Borges and Marco Tulio Valente. "What's in a GitHub star? understanding repository starring practices in a social coding platform". In: *Journal of Systems and Software* 146 (Dec. 2018), pp. 112–129. URL: `https://doi.org/10.1016/j.jss.2018.09.016` (cit. on p. 28).

[Cat+07] Marcelo Cataldo et al. "On coordination mechanisms in global software development". In: *International Conference on Global Software Engineering (ICGSE 2007).* IEEE. Sept. 2007, pp. 71–80. ISBN: 978-0-7695-2920-2. DOI: `10.1109/ICGSE.2007.33` (cit. on p. 13).

[Cha98] Dave Chaffey. *Groupware, Workflow and Intranets: Re-engineering the Enterprise with Collaborative Software.* Gulf Professional Publishing, 1998 (cit. on p. 8).

[CI14] Hui-Jung Chang and Wan-Zheng Ian. "Instant Messaging usage and interruptions in the workplace". In: *International Journal of Knowledge Content Development & Technology* 4.2 (Dec. 2014), p. 25. URL: `http://dx.doi.org/10.5865/IJKCT.2014.4.2.025` (cit. on p. 12).

[Dab+12] Laura Dabbish et al. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository". In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work.* CSCW '12. Seattle, Washington, USA: Association for Computing Machinery, Feb. 2012, pp. 1277–1286. ISBN: 9781450310864. DOI: `10.1145/2145204.2145396`.

URL: https://doi.org/10.1145/2145204.2145396 (cit. on pp. 22–24, 28).

[Dau16] Robin Daugherty. "A Branching and Releasing Strategy That Fits GitHub Flow". In: *Hackernoon* (Feb. 2016). URL: https://hackernoon.com/a-branching-and-releasing-strategy-that-fits-github-flow-be1b6c48eca2 (cit. on p. 14).

[Ern17] Michael Ernst. "How to create and review a GitHub pull request". In: *Computer Science & Engineering, University of Washington* (Mar. 2017). URL: https://homes.cs.washington.edu/~mernst/advice/github-pull-request.html (cit. on p. 9).

[Fri+02] Peter Fritzson et al. "The open source Modelica project". In: *Proceedings of The 2th International Modelica Conference*. 2002, pp. 18–19 (cit. on p. 64).

[GPD14] Georgios Gousios, Martin Pinzger, and Arie van Deursen. "An Exploratory Study of the Pull-Based Software Development Model". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 345–355. ISBN: 9781450327565. DOI: 10.1145/2568225.2568260. URL: https://doi.org/10.1145/2568225.2568260 (cit. on pp. 2, 9, 18, 23, 24).

[GHM98] John Grundy, John Hosking, and Warwick B Mugridge. "Inconsistency management for multiple-view software development environments". In: *IEEE Transactions on Software Engineering* 24.11 (Nov. 1998), pp. 960–981. ISSN: 0098-5589. DOI: 10.1109/32.730545 (cit. on p. 10).

[GPS04] Carl Gutwin, Reagan Penner, and Kevin Schneider. "Group Awareness in Distributed Software Development". In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. CSCW '04. Chicago, Illinois, USA: Association for Computing Machinery, Nov. 2004, pp. 72–81. ISBN: 1581138105. DOI: 10.1145/1031607.1031621. URL: https://doi.org/10.1145/1031607.1031621 (cit. on pp. 22, 24).

[Guz+13] Anja Guzzi et al. "Communication in open source software development mailing lists". In: *Proceedings of the 10th Working Conference on Mining Software Repositories.* IEEE Press. Oct. 2013, pp. 277–286. ISBN: 978-1-4799-0345-0. DOI: 10.1109/MSR.2013.6624039 (cit. on pp. 21, 24).

[HM19] Martine Haas and Mark Mortensen. "The Secrets of Great Teamwork". In: *Harvard Business Review* (Mar. 2019). URL: https://hbr.org/2016/06/the-secrets-of-great-teamwork (cit. on p. 14).

[Han19] Git Handbook. "Understanding the Git flow". In: *Waydev #1 Git Analytics Platform for Engineering Productivity* (Sept. 2019). URL: https://waydev.co/understanding-the-git-flow/ (cit. on p. 11).

[HL10] Lile Hattori and Michele Lanza. "Syde: A Tool for Collaborative Software Development". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2.* ICSE '10. Cape Town, South Africa: Association for Computing Machinery, May 2010, pp. 235–238. ISBN: 9781605587196. DOI: 10.1145/1810295.1810339. URL: https://doi.org/10.1145/1810295.1810339 (cit. on p. 22).

[Hel18] Heikki Hellgren. "Communicating in software development". In: *Hackernoon Newsletter* (Mar. 2018). URL: https://hackernoon.com/communicating-in-software-development-f3434c52eb23 (cit. on p. 1).

[HLT09] Konrad Hinsen, Konstantin Läufer, and George Thiruvathukal. "Essential Tools: Version Control Systems". In: *Computing in Science & Engineering* 11

(Nov. 2009), pp. 84–90. URL: https://doi.org/10.1109/MCSE.2009.194 (cit. on p. 10).

[Inj18] Injection-Marketing. “Communication can Save you Money and Time”. In: *Milacron Blog* (Feb. 2018). URL: https://www.milacron.com/mblog/2018/02/14/communication-can-save-you-money-and-time/ (cit. on p. 7).

[Jia+17] Jing Jiang et al. “Why and how developers fork what from whom in GitHub”. In: *Empirical Software Engineering* 22.1 (May 2017), pp. 547–578. ISSN: 1382-3256. URL: https://doi.org/10.1007/s10664-016-9436-6 (cit. on pp. 8, 11).

[JAM17] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. “Evolutionary trends of developer coordination: A network approach”. In: *Empirical Software Engineering* 22.4 (Nov. 2017), pp. 2050–2094. ISSN: 1382-3256. DOI: https://doi.org/10.1007/s10664-016-9478-9 (cit. on p. 1).

[JJ91] Peter Johnson-Lenz and Trudy Johnson-Lenz. “Post-mechanistic groupware primitives: rhythms, boundaries and containers”. In: *International Journal of Man-Machine Studies* 34.3 (Mar. 1991), pp. 395–417. URL: https://doi.org/10.1016/0020-7373(91)90027-5 (cit. on p. 8).

[Kal+14a] Eirini Kalliamvakou et al. “The code-centric collaboration perspective: Evidence from github”. In: *The Code-Centric Collaboration Perspective: Evidence from Github, Technical Report DCS-352-IR, University of Victoria* (2014). DOI: 10.1.1.728.2463 (cit. on p. 17).

[Kal+14b] Eirini Kalliamvakou et al. “The Promises and Perils of Mining GitHub”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 92–101. ISBN: 9781450328630. DOI: 10.1145/2597073.2597074. URL: https://doi.org/10.1145/2597073.2597074 (cit. on pp. 17, 18, 28).

[Kal+16] Eirini Kalliamvakou et al. "An In-Depth Study of the Promises and Perils of Mining GitHub". In: *Empirical Softw. Engg.* 21.5 (Oct. 2016), pp. 2035–2071. ISSN: 1382-3256. DOI: 10.1007/s10664-015-9393-5. URL: https://doi.org/10.1007/s10664-015-9393-5 (cit. on p. 11).

[KWC94] Sara Kiesler, Douglas Wholey, and Kathleen M Carley. "Coordination as linkage: The case of software development teams". In: *Organizational linkages: Understanding the productivity paradox* 52 (1994), pp. 96–123 (cit. on p. 14).

[KR19] Joe Kochitty and Rajshekhar Ratrey. "Barriers of Communication: Types of Barriers to Effective Communication". In: *Toppr* (Oct. 2019). URL: https://www.toppr.com/guides/business-correspondence-and-reporting/communication/barriers-in-communication/ (cit. on p. 12).

[Kum18] Krishna Kumar. "Learn The Three Different Types Of Version Control Systems". In: *Eduonix Blog* (Mar. 2018). URL: https://blog.eduonix.com/software-development/learn-three-types-version-control-systems/ (cit. on p. 10).

[Lan+10] Filippo Lanubile et al. "Collaboration tools for global software engineering". In: *IEEE software* 27.2 (Feb. 2010), pp. 52–55. ISSN: 0740-7459. DOI: 10.1109/MS.2010.39 (cit. on pp. 12, 13).

[LVD06] Thomas D. LaToza, Gina Venolia, and Robert DeLine. "Maintaining Mental Models: A Study of Developer Work Habits". In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. Shanghai, China: Association for Computing Machinery, May 2006, pp. 492–501. ISBN: 1595933751. DOI: 10.1145/1134285.1134355. URL: https://doi.org/10.1145/1134285.1134355 (cit. on pp. 21, 24).

[Le+15] Tien-Duy B. Le et al. “RCLinker: Automated Linking of Issue Reports and Commits Leveraging Rich Contextual Information”. In: *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension.* ICPC ’15. Florence, Italy: IEEE Press, May 2015, pp. 36–47 (cit. on pp. 24, 48).

[LLH16] Jing Liu, Jiahao Li, and Lulu He. “A comparative study of the effects of pull request on github projects”. In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).* Vol. 1. IEEE. Aug. 2016, pp. 313–322. ISBN: 978-1-4673-8846-7. DOI: 10.1109/COMPSAC.2016.27 (cit. on pp. 23, 24, 28).

[Mah95] A Mahon. “Groupware-Communication, Collaboration and Coordination”. In: *Intranet Journal* (1995). URL: https://web.archive.org/web/20110713051554/http://www.intranetjournal.com/faq/lotusbible.html (cit. on p. 8).

[MKN09] Dominique Matter, Adrian Kuhn, and Oscar Nierstrasz. “Assigning bug reports using a vocabulary-based expertise model of developers”. In: *2009 6th IEEE international working conference on mining software repositories.* IEEE. June 2009, pp. 131–140. ISBN: 978-1-4244-3493-0. DOI: 10.1109/MSR.2009.5069491 (cit. on p. 17).

[Mer15] Ines Mergel. “Open collaboration in the public sector: The case of social coding on GitHub”. In: *Government Information Quarterly* 32.4 (Oct. 2015), pp. 464–472. URL: https://doi.org/10.1016/j.giq.2015.09.004 (cit. on p. 17).

[NYY10] Kumiyo Nakakoji, Yunwen Ye, and Yasuhiro Yamamoto. “Supporting expertise communication in developer-centered collaborative software development environments”. In: *Collaborative Software Engineering*. Springer, Feb. 2010, pp. 219–236. ISBN: 978-3-642-10293-6. URL: `https://doi.org/10.1007/978-3-642-10294-3_11` (cit. on pp. 22, 24).

[Nes19] Anna Nesmiyanova. “Effective Collaboration Is the Secret to Your Software Development Project Success”. In: *Steel Kiwi* (Mar. 2019). URL: `https://steelkiwi.com/blog/collaboration-is-a-key-to-project-success/` (cit. on pp. 7, 8).

[Nii11] Tuomas Niinimaki. “Face-to-face, email and instant messaging in distributed agile software development project”. In: *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*. IEEE. Nov. 2011, pp. 78–84. ISBN: 978-1-4577-1839-7. DOI: `10.1109/ICGSE-W.2011.15` (cit. on p. 12).

[NPL09] Tuomas Niinimaki, Arttu Piri, and Casper Lassenius. “Factors affecting audio and text-based communication media choice in global software development projects”. In: *2009 Fourth IEEE International Conference on Global Software Engineering*. IEEE. Aug. 2009, pp. 153–162. ISBN: 978-0-7695-3710-8. DOI: `10.1109/ICGSE.2009.23` (cit. on p. 13).

[Pan+14] Sebastiano Panichella et al. “How developers’ collaborations identified from different sources tell us about code changes”. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE. Dec. 2014, pp. 251–260. ISBN: 978-1-4799-6146-7. DOI: `10.1109/ICSME.2014.47` (cit. on pp. 21, 24).

[Per+16] Yasset Perez-Riverol et al. *Ten simple rules for taking advantage of Git and GitHub*. July 2016. DOI: `10.1371/journal.pcbi.1004947` (cit. on p. 11).

[Pip15] Achilleas Pipinellis. *GitHub essentials*. Packt Publishing Ltd, 2015 (cit. on p. 9).

[RR14] Mohammad Masudur Rahman and Chanchal K. Roy. "An Insight into the Pull Requests of GitHub". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 364–367. ISBN: 9781450328630. DOI: `10.1145/2597073.2597121`. URL: `https://doi.org/10.1145/2597073.2597121` (cit. on pp. 2, 10).

[RK10] Cobra Rahmani and Deepak Khazanchi. "A study on defect density of open source software". In: *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*. IEEE. Sept. 2010, pp. 679–683. ISBN: 978-1-4244-8198-9. DOI: `10.1109/ICIS.2010.11` (cit. on p. 17).

[RR00] Pierre N Robillard and Martin P Robillard. "Types of collaborative work in software engineering". In: *Journal of Systems and Software* 53.3 (Sept. 2000), pp. 219–224. URL: `https://doi.org/10.1016/S0164-1212(00)00013-3` (cit. on p. 12).

[RC16] Margaret Rouse and Meredith Courtemanche. "What is GitHub? - Definition from WhatIs.com". In: *SearchITOperations* (2016). URL: `https://searchitoperations.techtarget.com/definition/GitHub` (cit. on p. 11).

[SB97] Carolyn B Seaman and Victor R Basili. "Communication and organization in software development: an empirical study". In: *IBM Systems Journal* 36.4 (1997), pp. 550–563. ISSN: 0018-8670. DOI: `10.1147/sj.364.0550` (cit. on pp. 23, 24).

[SRP17] Todd Sedano, Paul Ralph, and Cécile Péraire. "Software Development Waste". In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Buenos Aires, Argentina: IEEE Press, May 2017, pp. 130–140. ISBN: 9781538638682. DOI: `10.1109/ICSE.2017.20`. URL: `https://doi.org/10.1109/ICSE.2017.20` (cit. on p. 1).

[She+14] Jyoti Sheoran et al. "Understanding Watchers on GitHub". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 336–339. ISBN: 9781450328630. DOI: `10.1145/2597073.2597114`. URL: `https://doi.org/10.1145/2597073.2597114` (cit. on p. 9).

[SS15] Janet Siegmund and Jana Schumann. "Confounding parameters on program comprehension: a literature survey". In: *Empirical Software Engineering* 20.4 (May 2015), pp. 1159–1192. ISSN: 1382-3256. URL: `https://doi.org/10.1007/s10664-014-9318-8` (cit. on p. 63).

[Sin+13] Leif Singer et al. "Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators". In: *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. CSCW '13. San Antonio, Texas, USA: Association for Computing Machinery, Feb. 2013, pp. 103–116. ISBN: 9781450313315. DOI: `10.1145/2441776.2441791`. URL: `https://doi.org/10.1145/2441776.2441791` (cit. on pp. 23, 24).

[Sin13] Sandeep Singh. "Analysis of bug tracking tools". In: *International Journal of Scientific & Engineering Research* 4.7 (July 2013), p. 134. ISSN: 2229-5518 (cit. on p. 17).

[Sou+04] Cleidson R. B. de Souza et al. "How a Good Software Practice Thwarts Collaboration: The Multiple Roles of APIs in Software Development". In:

SIGSOFT Softw. Eng. Notes 29.6 (Oct. 2004), pp. 221–230. ISSN: 0163-5948. DOI: 10.1145/1041685.1029925. URL: https://doi.org/10.1145/1041685.1029925 (cit. on p. 1).

[Sto+14] Margaret-Anne Storey et al. "The (R) Evolution of Social Media in Software Engineering". In: *Proceedings of the on Future of Software Engineering.* FOSE 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 100–116. ISBN: 9781450328654. DOI: 10.1145/2593882.2593887. URL: https://doi.org/10.1145/2593882.2593887 (cit. on pp. 21, 22, 24).

[Sto+17] Margaret-Anne Storey et al. "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development". In: *IEEE Trans. Softw. Eng.* 43.2 (Feb. 2017), pp. 185–204. ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2584053. URL: https://doi.org/10.1109/TSE.2016.2584053 (cit. on pp. 24, 64).

[Thi+07] M. Rita Thissen et al. "Communication Tools for Distributed Software Development Teams". In: *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce.* SIGMIS CPR '07. St. Louis, Missouri, USA: Association for Computing Machinery, Apr. 2007, pp. 28–35. ISBN: 9781595936417. DOI: 10.1145/1235000.1235007. URL: https://doi.org/10.1145/1235000.1235007 (cit. on pp. 8, 12, 14).

[TDH14a] Jason Tsay, Laura Dabbish, and James Herbsleb. "Influence of Social and Technical Factors for Evaluating Contribution in GitHub". In: *Proceedings of the 36th International Conference on Software Engineering.* ICSE 2014. Hyderabad, India: Association for Computing Machinery, May 2014, pp. 356–

366. ISBN: 9781450327565. DOI: `10.1145/2568225.2568315`. URL: `https://doi.org/10.1145/2568225.2568315` (cit. on pp. 23, 24, 28).

[TDH14b] Jason Tsay, Laura Dabbish, and James Herbsleb. "Let's Talk about It: Evaluating Contributions through Discussion in GitHub". In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: Association for Computing Machinery, Nov. 2014, pp. 144–154. ISBN: 9781450330565. DOI: `10.1145/2635868.2635882`. URL: `https://doi.org/10.1145/2635868.2635882` (cit. on p. 9).

[Vas+15] Bogdan Vasilescu et al. "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, Aug. 2015, pp. 805–816. ISBN: 9781450336758. DOI: `10.1145/2786805.2786850`. URL: `https://doi.org/10.1145/2786805.2786850` (cit. on p. 9).

[Ver20] Arun Kumar Verma. Feb. 2020. URL: `https://sites.google.com/view/master-thesis-verma-20/` (cit. on pp. 5, 67).

[Yu+15] Yue Yu et al. "Wait for It: Determinants of Pull Request Evaluation Latency on GitHub". In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. MSR '15. Florence, Italy: IEEE Press, May 2015, pp. 367–371. ISBN: 9780769555942 (cit. on pp. 9, 28).

[Yu+16] Yue Yu et al. "Reviewer Recommendation for Pull-Requests in GitHub". In: *Inf. Softw. Technol.* 74.C (June 2016), pp. 204–218. ISSN: 0950-5849. DOI: `10.1016/j.infsof.2016.01.004`. URL: `https://doi.org/10.1016/j.infsof.2016.01.004` (cit. on p. 9).

[Zha+18] Yang Zhang et al. “Within-Ecosystem Issue Linking: A Large-Scale Study of Rails”. In: *Proceedings of the 7th International Workshop on Software Mining*. SoftwareMining 2018. Montpellier, France: Association for Computing Machinery, 2018, pp. 12–19. ISBN: 9781450359757. DOI: `10.1145/3242887.3242891`. URL: `https://doi.org/10.1145/3242887.3242891` (cit. on p. 47).

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeitin gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 15. März 2020

Arun Kumar Verma