Impact of Using Suggestion Bot While Code Reviewing

Nivishree Palvannan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Chris Brown, Chair
Na Meng
Muhammad Ali Gulzar

May 9, 2023
Blacksburg, Virginia

Keywords: Pull Request, GitHub, Code Review, Suggested Changes

Impact of Using Suggestion Bot While Code Reviewing

Nivishree Palvannan

(ABSTRACT)

Peer code reviews play a critical role in maintaining code quality, and GitHub has introduced several new features to assist with the review process. One of these features is *suggested changes*, which allows for precise code modifications in pull requests to be suggested in review comments. Despite the availability of such helpful features, many pull requests remain unattended due to lower priority. To address this issue, we developed a bot called "Suggestion Bot" to automatically review the codebase using GitHub's suggested changes functionality. An empirical study was also conducted to compare the effectiveness of this bot with manual reviews. The findings suggest that implementing this bot can expedite response times and improve the quality of pull request comments for pull-based software development projects. In addition to providing automated suggestions, this feature also offers valuable, concise, and targeted feedback.

Impact of Using Suggestion Bot While Code Reviewing

Nivishree Palvannan

(GENERAL AUDIENCE ABSTRACT)

Code review, often known as peer review, is a process used to ensure the quality of software.Code review is a process in software development that involves one or more individuals examining the source code of a program, either after it has been implemented or during a pause in the development process. The creator of the code cannot be one of the individuals. “Reviewers” refers to the individuals conducting the checking, excluding the author. However, the majority of reviewers won't have the time to examine and validate the peer's code base, so they'll assign it the lowest priority possible. This could cause pull requests to stall out without being reviewed. Therefore, as part of our research, we are creating a bot called SUGGESTION BOT that provides code changes in pull requests. The author can then accept, reject, or alter these ideas as a necessary component of the pull request. Additionally, we compared the effectiveness of our bot with the manual pull request review procedure, which clearly demonstrated that the incorporation of this bot significantly shortened the turnaround time. Besides giving automated recommendations, this functionality also provides useful, brief, and focused feedback.

Dedication

To my loving family,

whose unwavering support and encouragement

have been the driving force behind my academic achievements.

To my advisor,

for his expert guidance, mentorship, and patience

throughout the research process

And to my dear friends,

for their constant motivation, camaraderie,

and belief in my abilities.

Acknowledgments

I would like to express my deepest gratitude to the following individuals who have contributed significantly to the completion of my Master's thesis:

First and foremost, I am immensely grateful to my advisor, Dr. Chris Brown, for his expert guidance, unwavering support, and invaluable mentorship throughout the research process. Their insightful feedback, constructive criticism, and patience have been instrumental in shaping this thesis.

I am also grateful to the members of my thesis committee, Dr. Na Meng and Dr. Muhammad Ali Gulzar, for their valuable inputs, suggestions, and critical evaluation of my work. Their expertise and feedback have immensely enriched the quality of this thesis.

I extend my heartfelt thanks to my family for their unwavering love, support, and encouragement. Your constant belief in my abilities and understanding of my academic pursuits have been a driving force behind my achievements.

I am grateful to my friends and colleagues for their camaraderie, motivation, and intellectual discussions, which have enriched my research experience and made this journey more enjoyable.

Finally, I express my gratitude to all the participants who generously contributed their time and insights to my research, without whom this thesis would not have been possible.

I am deeply grateful to all of you for your support, encouragement, and contributions to my Master's thesis. Thank you for being an integral part of my academic journey

Contents

List of Figures **viii**

List of Tables **ix**

1 Introduction **1**

2 Background **7**

2.1 GitHub Pull requests 7

2.2 Factors influencing pull requests turnaround time 9

2.3 Automated Tool for Code Review Process 10

3 Suggestion Bot **13**

3.1 Overview of Suggestion Bot 13

3.2 Implementation of Suggestion Bot 16

3.3 Working of Suggestion Bot 18

4 Related Work **20**

4.1 Empirical studies on the using tools for code reviewing 20

4.2 Empirical studies on the existing automated tools and bots 22

5 Study Methodology **25**

5.1 Data Collection 26

5.2 Data Analysis 28

5.3 Ethical Considerations 30

6 Results **32**

6.1 RQ1: Usefulness of the Suggestion Bot 34

6.2 RQ2: How quickly are pull requests are reviewed using Suggestion Bot . . . 39

6.3 RQ3: Impacts in using a bot while reviewing pull requests 41

7 Discussion **45**

7.1 Summary of Findings 45

7.2 Limitations 46

7.3 Suggestions for Future Research 48

8 Conclusions **50**

Bibliography **51**

Appendices **54**

Appendix A Survey **55**

A.1 Pre-Survey 55

A.2 Post-Survey 56

List of Figures

2.1 An image of a GitHub pull request 9

3.1 An image of a GitHub suggestions from the Bot 16

6.1 Time difference between manual review and by using Suggestion Bot 39

List of Tables

6.1 Factors responsible for usefulness of the Suggestion Bot 34

List of Abbreviations

KMO Kaiser-Meyer-Olkin

p-value Probability value

PCA Principal Component Analysis

PR Pull Request

A pull request is a feature commonly used in software development workflows using version control systems, such as Git, to propose changes to a codebase. It is also known as a merge request in some version control system

Kaiser-Meyer-Olkin (KMO) is a statistic used in the field of statistics and data analysis to assess the sampling adequacy for conducting factor analysis or principal component analysis. It is a measure of how suitable the data is for these multivariate techniques, specifically in terms of their applicability for extracting underlying factors or components from the data.

Principal Component Analysis (PCA) is a multivariate statistical technique used for data analysis and dimensionality reduction. It involves transforming a set of correlated variables into a new set of uncorrelated variables, known as principal components, which capture the most important information in the original data.

In statistics, a p-value is a measure of the evidence against a null hypothesis. It represents the probability of obtaining a result as extreme or more extreme than the observed result, assuming that the null hypothesis is true. The null hypothesis is typically a statement that there is no effect or relationship between two variables of interest.

Chapter 1

Introduction

Pull-based software development, as implemented by GitHub, has become a popular model for collaborating on distributed software development [7]. It offers several benefits, including branching and isolated development, efficient code reviews, and streamlined collaboration among team members. One key aspect of pull-based development is the ability to create branches for different features, bug fixes, or experiments. This allows developers to work independently on separate branches without directly modifying the main branch or interfering with each other's work. Each branch represents a specific set of changes or a feature, making it easier to manage and organize development efforts. When a developer is ready to share their changes with the team, they can create a *pull request*. The pull request serves as a formal request to review and discuss the changes before merging them into the main branch. This allows for a structured and collaborative review process where team members can provide feedback, suggestions, and comments on the proposed changes.

The code review process is a crucial part of pull-based development [22]. Reviewers can thoroughly inspect the changes made in the form of commits, review the code modifications, and provide feedback on the overall quality, functionality, and adherence to coding standards. This helps identify and fix potential issues, bugs, and improvements before the changes are merged into the main branch. Pull requests also facilitate discussions among team members. Reviewers and developers can have conversations about the proposed changes, clarifying intentions, addressing questions, and discussing potential alternatives. This promotes col-

laboration and ensures that the changes align with the overall goals and requirements of the project [16]. The pull request process typically involves getting approval from project maintainers or peer workers before changes are merged into the main repository. Once the changes are approved, they can be merged into the main branch, making them part of the official codebase. This helps maintain code quality, consistency, and stability, as changes are thoroughly reviewed and tested before being merged. GitHub, in particular, has provided a user-friendly and feature-rich platform for implementing pull-based development. It offers a range of tools for creating, managing, and reviewing pull requests, such as inline comments, code review templates, and integration with popular continuous integration and deployment tools. These features have made pull requests a widely adopted method for collaborating on software development projects, both open-source and closed-source, with increased visibility, accountability, and efficiency.

Despite the benefits of pull requests, such as improved collaboration and code quality, they can sometimes suffer from stagnation. There are several common factors that can contribute to pull requests becoming stagnant, including the reviewing task being deprioritized and unclear comments from reviewers, which can lead to delays in the review and approval process. One factor that can contribute to the stagnation of pull requests is when reviewing tasks are deprioritized [4]. In a busy development environment, reviewers may have competing priorities, such as urgent bug fixes or other tasks, which can cause delays in reviewing pull requests. This can result in pull requests sitting idle for extended periods, leaving the author of the pull request waiting for feedback and approval. Another factor that can lead to stagnation is unclear comments from reviewers [10]. Reviewers may leave comments that are not detailed or specific enough, making it challenging for the author of the pull request to understand the feedback and take appropriate action to address the comments. This can lead to back-and-forth discussions, delays in making necessary changes, and ultimately, the

pull request remaining stagnant. In some cases, the lack of clear guidelines or documentation on the review process or coding standards can also contribute to stagnation. If there are no established processes or standards for reviewing pull requests, it can result in confusion, inconsistency, and delays in the review process.

Bots and automated tools can be useful in supporting the reviewing process of pull requests by automating predefined and repetitive tasks, such as code formatting, linting, and automated testing. These tools can help ensure that code contributions meet the established coding standards, reduce human error, and save time for reviewers and contributors. However, it's true that some bots can sometimes interrupt the developer workflow and create challenges in the pull request review process [21]. One common issue is the verbosity of comments and explanations provided by the existing bots. Verbose comments can be overwhelming and difficult for the author of the pull request to understand. Lengthy comments can make it challenging to identify the specific issue or suggested improvement, and can even create confusion or frustration. This can result in delays in addressing the feedback, and may lead to misunderstandings or misinterpretations of the required changes. One possible reason for verbose comments from bots is that they may be configured to provide detailed feedback on various aspects of the code, including minor issues or nitpicks. While this level of detail can be helpful in some cases, it can also be overwhelming and time-consuming for the author of the pull request, especially when dealing with multiple comments from multiple bots. Another issue with verbose comments from bots is that they may not provide sufficient context or explanations, making it difficult for the author of the pull request to understand the rationale behind the feedback. This can lead to confusion and uncertainty about how to address the comments, resulting in additional back-and-forth discussions and delays in the review process.

To overcome all these problems, we developed a bot called SUGGESTION BOT to address

the challenges of pull request reviewing, with a focus on reducing the turnaround time. Leveraging the features of the Black tool [3] and GitHub's suggested changes feature SUGGESTION BOT provides comments as suggestions directly in the pull request, instead of verbose statements. Black is a Python code formatter that automatically formats your code according to a set of rules. It is designed to produce consistently formatted code that is easy to read and follow. Using Black can help you maintain a consistent coding style throughout your project, and can also save time by automatically formatting your code instead of manually formatting it. This allows the author to easily see and accept or reject the suggested code changes. Our empirical study aims to investigate the impact of using automated bots for pull request reviewing compared to manual review processes, focusing on three research questions.

RQ1 How useful is recommendation of suggested changes using a bot while reviewing pull requests?

A pull request is a step for merging code to the main branch where developers notify their team members and use it for code review discussion. However, slow review tasks can cause delays and merge conflicts. Suggested changes feature of GitHub can help by providing concise feedback instead of verbose comments, and a SUGGESTION BOT can also streamline the code review process. These tools can be beneficial for a developer's day-to-day activities.

RO2 How quickly are pull requests are reviewed using a bot?

Many developers avoid reviewing pull request due to variety of reason. Many developers don't believe they're skilled enough to be able to review their teammate's code. In many cases, code review skills are acknowledged almost as strongly as the quality of code a developer will produce. Some developers avoid reviewing if they feel some unfamiliarity with the changes committed in the pull request. They feel developers who participated in the discussions should review the pull request. One more major reason which delays a pull requests is

prioritization of the reviewing work by the developers. So this finding on how quickly pull requests are reviewed using SUGGESTION BOT will help the team to analyze the usage of bot while reviewing pull requests and to reduce the turnaround time of it [5].

RQ3 What are the impacts of using SUGGESTION BOT while reviewing pull requests?

Using SUGGESTION BOT in the pull request reviewing process can provide significant advantages. The bot can offer code changes as inline comments in pull requests, allowing authors to easily accept, reject, or edit these suggestions as part of the code review process. This can lead to reduced turnaround time for pull requests and overall time savings in the product development cycle, while also mitigating potential merge conflicts. The findings of this study provide an overview of the positive impacts and benefits that a SUGGESTION BOT can bring to the reviewing process, aiding the team's integration with the development cycle.

To analyze the usage and impact of SUGGESTION BOT, we conducted a user study and semi-structured interviews with 25 participants who had prior experience with software development and GitHub. We compared and contrasted the advantages of manual review versus review using the bot. The results of the study clearly indicated that when an automated tool or bot is used for the reviewing process and provides suggestions in a clear and understandable manner, it increases the turnaround time of pull requests and results in faster merging of the pull requests into the master branch. One of the key findings of the study was that the use of the SUGGESTION BOT significantly reduced the time spent on reviewing and addressing comments in pull requests. The suggestions provided by the SUGGESTION BOT were clear and easy to understand, which helped the authors of the pull requests quickly make the necessary changes and resubmit their updates. This resulted in faster iterations and reduced delays in the review process, ultimately leading to quicker merging of the pull requests into the main repository. Moreover, participants also reported

that the use of the SUGGESTION BOT improved the quality of the code reviews. The automated suggestions provided by the SUGGESTION BOT were precise and focused on the specific changes needed, which helped in identifying and fixing issues more effectively. This resulted in fewer back-and-forth iterations of reviews and reduced the chances of missing important feedback.

By addressing the suggested changes in a timely manner, the pull requests were more likely to be merged smoothly without conflicts arising during the integration process. This helped in streamlining the development workflow and avoiding delays caused by resolving merge conflicts. Overall, the findings of the study highlighted the positive impact of using a SUGGESTION BOT in the pull request reviewing process. The clear and understandable suggestions provided by the bot improved the turnaround time of pull requests, enhanced the quality of code reviews, and reduced the chances of merge conflicts. These advantages can significantly contribute to a more efficient and streamlined development cycle, ultimately leading to faster integration of changes into the main repository.

Chapter 2

Background

Social coding platforms like GitHub are widely utilized for software development, and bots are frequently used to speed up the development process [18]. These bots communicate with human developers by leaving comments on the platforms' various discussion interfaces. The evaluation of pull requests (PRs) and the usefulness of bots in the code review process will be one of the primary interaction types we will focus on. PRs are a popular way to propose changes to a codebase [16], and they go through a review process in which human reviewers and software development bots evaluate the changes for quality, functionality, and adherence to coding standards. In code review, software development bots are used to automate repetitive processes and provide comments on suggested modifications. They can review modifications based on preset rules, guidelines, and best practices, and conduct duties including validating code formatting, identifying potential problems, verifying test cases, and assuring coding standards compliance.

2.1 GitHub Pull requests

The well-known web-based source code hosting site GitHub provides software professionals with a practical way to collaborate on the construction of open source software. GitHub employs a pull-based development paradigm, in which any user can make changes to any public project [9]. Individual developers or organizations can own projects on GitHub. Develop-

ers who want to contribute can create new repositories or fork existing ones, then continue working. GitHub independently maintains the source code and related materials, such as committed code and commit comments, for base and forked projects [18]. The developers will then add a new feature or make changes to the existing code in their local branch. Pull requests are used to merge code changes from the local branch to the master branch. Pull requests are development procedures that developers use to push and incorporate code into the master or main branch [16]. A pull request is a GitHub action in which a contributor requests that the repository's maintainer review code that they want to incorporate into a project [16]. Before being merged into the master branch, the code alterations will be checked and reviewed by the reviewer.

In the past, code updates were reviewed through inspection, which included formal meetings in which all participants had to be in the same place at the same time [5]. However, the existing code review method via pull requests is asynchronous and accommodates geographically dispersed reviewers. The modern review process is a tool-based and informal code review process that is an effective strategy for gathering all of the information required for that code change in a single pull request, and communication about the code changes in the pull requests is context-specific. As part of the code review process, the authors of pull requests can subsequently apply or reject the code modifications suggested by collaborators and reviewers through comments. As a result, pull requests rely on peer code reviews, which raises a number of challenges. Peer code review, which is a manual inspection of source code by developers other than the author, is recognized as a valuable approach for minimizing software faults and enhancing software project quality. The bulk of pull requests merely modify a few dozen lines of code, yet due to merge conflicts caused by pull request stagnation, the most of them are not merged to the source branch [16]. Despite having numerous advantages, such as fast turnaround, expanded options for community interaction, and de-

creased time to absorb contributions, the pull request model has its own drawbacks, which result in merge disputes and pull request stagnation.

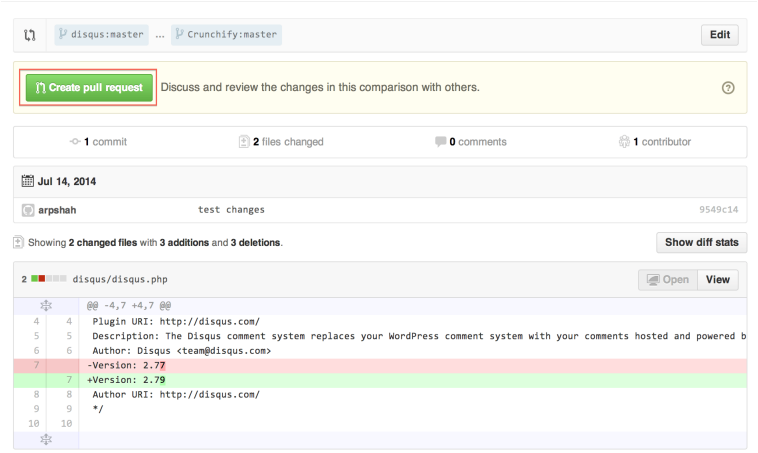

Figure 2.1: An image of a GitHub pull request

2.2 Factors influencing pull requests turnaround time

Modern software development workflows routinely include code reviews due to their numerous advantages [17]. As it necessitates the involvement of multiple people, code review is frequently the most time-consuming component of code integration activities. The reviewers' comments on a code's improvement and long-term maintainability make up over 50 of all comments. Code improvements are comments or changes about code in terms of readability, commenting, consistency, dead code removal, etc., but do not involve correctness or defects. Programmers ranked code improvement as a main motivation to code review process. Despite the fact that code review has many benefits, including the capacity to discover errors and assure long-term maintainability of the code, it also has its own drawbacks. One of the most frequently raised issue on code reviewing is difficult in understanding the comments given by reviewer. The style of inline comments will vary between reviewers and most of

the times that author of the pull request are not clear of those comments which serves as an important factor in influencing the pull request merge time.

Modern code reviews process are very expensive. Developers spend on average six hours a week reviewing changes of the others [11]. And coder reviews need to performed by the people with specific set of skills and the reviewers who are familiarize with the code base will review it faster than the reviewers who are not familiar and need to spend some time in learning about the features and changes incorporated in the pull request. Since this is a time consuming process most of the reviewer de-prioritize [19] the code review task and reviewers refuses to take ownership of the task which results in the stagnation of the pull requests.

The effect of gender bias is largely visible in GitHub and it also affect the merge time. Women face more resistance to merging pull requests to the master branch than men. In their study on gender bias on GitHub by Nasif Imtiaz [8] found that Women's pull requests generate more discussion, receive more change suggestions, and take more time to get accepted and merged to the source branch. So gender bias on GitHub also serves as an hidden factor in increasing the merge time pull requests [8].

2.3 Automated Tool for Code Review Process

Bots and automated tools have emerged as efficient and effective ways to automate various tasks in software development [14]. They can reduce the workload and effort required, as they operate automatically without needing manual intervention from a human user every time. This can be particularly beneficial in code review processes, where bots can assist in addressing biased behavior and streamline repetitive tasks. Code review often involves providing feedback on topics such as code refactoring, team norms, typos, static and dynamic errors, and identifying dead code. Bots can be utilized by project maintainers to lighten their

workload in these areas [20]. For example, bots can verify if contributions comply with a specified style guide, check if the code compiles and passes tests, fix bugs, propose tools, update dependencies, and correct static analysis violations. These tasks, which can be time-consuming and tedious, can be simplified using bots or automated tools, allowing maintainers to focus on development and review activities and ultimately boost productivity while saving time and effort. However, despite the benefits, bots used in code review processes also face challenges. One common issue is the verbosity of bot comments, where they may provide excessive detail that can impede the development workflow and irritate developers. Bots may also perform an excessive number of actions or carry out unwanted or unasked-for jobs on pull requests, which can be perceived as noise by developers. Additionally, incorporating existing bots into existing projects may require prior knowledge of how the bots work and technical expertise in integrating and managing their workflow. One specific example of a code review bot is the "Review Bot" [5]. This bot automates the code review process by analyzing pull requests for code quality, style violations, and other best practices. However, like any other technology, code review bots also have some disadvantages, which may include:

False positives/negatives: Code review bots may sometimes generate false positives or false negatives, providing inaccurate feedback on code quality or missing potential issues. This can lead to confusion and may require additional effort from developers to verify and address the bot's feedback.

Lack of contextual understanding: Code review bots may lack contextual understanding of the specific requirements, goals, and constraints of a project. They may not be able to fully grasp the nuances of the codebase, project architecture, or development workflow, leading to generic or irrelevant feedback.

Over-reliance on automation: Code review bots can lead to over-reliance on automation, potentially reducing the thoroughness and critical thinking of human code reviewers. De-

velopers may rely solely on the bot's feedback without fully understanding the underlying issues or reasoning behind the suggestions, leading to missed opportunities for learning and improvement.

Impersonal feedback: Code review bots may provide feedback in a robotic and impersonal manner, lacking the human touch and empathy. This can sometimes lead to frustration or demotivation among developers, as they may prefer more human-like interactions during the code review process.

Limitations in customization: Code review bots may have limitations in customization, as they are pre-programmed with predefined rules or templates. Customizing the bot's behavior to suit specific project requirements may require technical expertise and additional effort, making it challenging for teams with limited resources or technical knowledge.

In conclusion, bots and automated tools offer significant advantages in automating tasks in software development, including code review processes [20]. They can reduce the burden on project maintainers and improve productivity. However, challenges such as verbosity, excessive actions, and integration complexities may also arise. Therefore, careful consideration and customization of bots to suit the specific needs of a project are necessary for successful implementation and efficient human-bot interaction in the code review process.

Chapter 3

Suggestion Bot

In order to improve the turnaround time of pull request and to overcome all the disadvantages of the bots discussed above we created a bot called SUGGESTION BOT that can communicate with GitHub APIs. The goal of this bot is to examine pull requests and speed up their turnaround time, saving developers time in the process.

3.1 Overview of Suggestion Bot

In a perfect working environment, the majority of pull requests are overlooked or avoided because of their low priority, which increases the number of merge conflicts [19]. Therefore, adding a code review and change SUGGESTION BOT will save time for both the reviewer and the author while also speeding up the pull request turnaround time. The SUGGESTION BOT is a tool that can be integrated with any repository, regardless of the programming language used, and the integration process is relatively simple and straightforward compared to existing pull request bots.

The SUGGESTION BOT serves as an automated assistant in the code review process, providing suggestions for code improvements based on predefined rules or templates. It analyzes the code changes in the pull request and generates comments with suggested changes, such as code refactoring, style violations, and other best practices. These suggestions can then be reviewed by the author of the pull request and accepted, rejected, or edited as needed.

One of the key advantages of the SUGGESTION BOT is its ease of integration into the existing workflow. Compared to other pull request bots, the integration step is minimal and straightforward, making it accessible for teams using different programming languages. The bot utilizes GitHub APIs to publish the suggested changes as comments in the pull request, allowing for easy collaboration and feedback exchange between the bot and the human reviewers.

The study focuses on comparing the time required to evaluate a pull request with and without the "SUGGESTION BOT." The main goal is to determine if the bot can expedite the code review process by automatically identifying and suggesting changes based on predefined rules or templates. This could potentially save time for human reviewers, who would then review and validate the SUGGESTION BOT's suggestions instead of manually identifying and suggesting changes themselves.

Additionally, the study aims to assess the effectiveness of the SUGGESTION BOT in terms of the quality of its suggestions. The SUGGESTION BOT's suggestions will be compared to those provided by human reviewers to evaluate the accuracy and relevance of the bot's suggestions in improving code quality and adherence to coding standards. To streamline the study and simplify its purposes, specific issues that can be addressed by both the bot and the participants have been considered. These issues include unused imports, camel casing issues, unwanted semicolons, and whitespace problems.

In summary, the SUGGESTION BOT is a tool that aids in the code review process, providing automated suggestions for code improvements. The study focuses on evaluating the time required to evaluate pull requests with and without the bot, as well as assessing the effectiveness and advantages of the bot in terms of its ease of integration, accuracy of suggestions, and potential time savings in the code review process. The study specifically uses the Python programming language and GitHub APIs for analyzing the effectiveness of the

SUGGESTION BOT in pull request code reviews.

Figure 2.2 illustrates an example of an inline comment provided by the SUGGESTION BOT during the code review process. The SUGGESTION BOT's comment is aimed at providing feedback on a specific section of code, highlighting potential areas for improvement or adherence to coding best practices.

The comment provided by the SUGGESTION BOT serves as a starting point for the author, who has the autonomy to accept, reject, or update the suggestion based on their requirements. This flexibility allows the author to exercise their judgment and make decisions that align with their coding style, project requirements, and overall vision.

If the author chooses to accept the suggestion, it indicates that they agree with the BOT's feedback and will incorporate the suggested changes into the code. This demonstrates a willingness to follow the coding best practices and standards recommended by the SUGGESTION BOT, enhancing the quality of the codebase.

On the other hand, if the author decides to reject the suggestion, it means that they do not find it applicable or necessary in the given context. This could be due to project-specific requirements, design considerations, or other factors. Rejecting a suggestion does not necessarily mean that the SUGGESTION BOT's feedback is invalid, but rather that the author has made a conscious decision to deviate from it based on their judgment.

Additionally, the author may choose to update the suggestion, indicating that they acknowledge the feedback but propose a different approach or solution. This reflects the author's critical thinking and ability to incorporate feedback while making informed decisions that align with their coding style and project requirements.

The ability to accept, reject, or update the suggestions provided by the SUGGESTION BOT empowers the author to exercise their expertise and judgment in the code review process.

It promotes a collaborative approach where the author and the SUGGESTION BOT work together to improve the code quality, while allowing for flexibility and customization based on the author's requirements. This iterative feedback loop between the SUGGESTION BOT and the author fosters a constructive code review process and encourages continuous improvement in the team's coding practices.

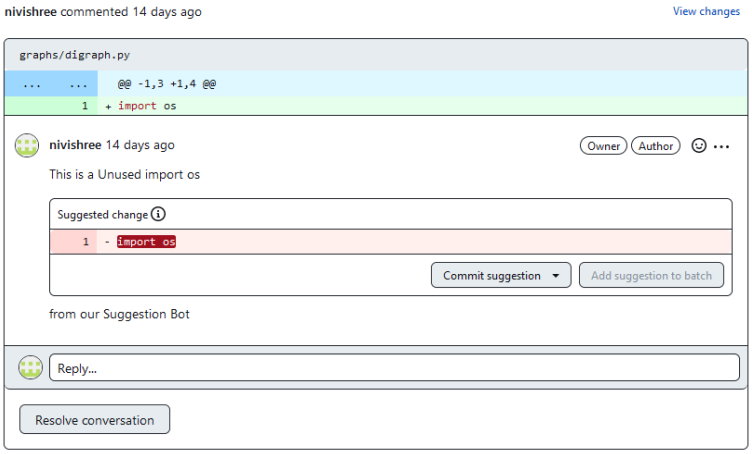

Figure 3.1: An image of a GitHub suggestions from the Bot

3.2 Implementation of Suggestion Bot

The SUGGESTION BOT leverages the power of GitHub's suggested changes feature and the Black tool to streamline the code review and formatting process. Here's an expanded description:

The Suggestion Bot is a tool that takes advantage of the suggested changes feature introduced by GitHub in its beta edition. This feature allows collaborators to propose code modifications directly within pull requests through inline comments. Instead of using external tools or performing copy-and-paste operations, the suggested changes feature enables the creation of code snippets that reflect the intended modifications.

The Suggestion Bot integrates with this feature, making it easier for collaborators to suggest code changes during the code review process. Collaborators can add inline comments to specific lines or sections of code, providing their suggestions for improvement or modification. These suggestions are presented as code snippets within the pull request itself, making it convenient for the pull request author and other collaborators to review and consider the proposed changes.

To enhance the code quality and maintain a consistent code style, the Suggestion Bot incorporates the Black tool. Black is an open-source, PEP 8 compliant code formatter with its own style. It can be used as a static code analysis tool for identifying static errors and ensuring code quality. The tool runs on various platforms such as Windows, Linux, macOS, and Azure through Docker.

When the Suggestion Bot detects code suggestions within the pull request, it can automatically apply the Black formatter to the suggested code snippets. This ensures that the suggested changes adhere to the agreed-upon coding style and improves the consistency, generality, and readability of the codebase. By reducing differences in code formatting (known as git diffs), the Suggestion Bot helps maintain a cleaner and more streamlined version control history.

The pull request author can review the suggested changes with the applied Black formatting and decide whether to accept, reject, or edit the suggestions. This integrated process eliminates the need for manual formatting and reduces the back-and-forth between collaborators during code reviews.

Overall, the Suggestion Bot enables a smoother and more efficient code review process by leveraging GitHub's suggested changes feature and integrating the power of the Black code formatter to ensure code quality and consistent formatting within the pull request workflow.

3.3 Working of Suggestion Bot

The suggestion bot's backend is built on a Python server that utilizes Flask for its implementation. On the other hand, the user interface of the suggestion bot is a simple interface developed using HTML, CSS, and Javascript. When a user interacts with the user interface, they provide specific details such as the repository name, repository owner, Github Token, and Pull request number. These inputs are crucial for the bot to perform its tasks effectively. Once the user submits these details, the data is sent to the backend Python server.

In the backend, the server uses both Github APIs and the black tool to analyze and review Pull requests based on user-provided information. The server interacts with the Github APIs to retrieve the relevant data associated with the specified Pull request and repository. It achieves this by fetching details such as the repository name, owner, and the latest commit information using the provided Pull request number.

After fetching the relevant data from the GitHub APIs, the server proceeds to clone the repository locally. This is done by utilizing the provided repository name and owner information. By cloning the repository, the server gains direct access to the code files involved in the Pull request, enabling further analysis. With the repository cloned, the server focuses on the specific files that have been modified as part of the Pull request. These modified files are identified based on the information extracted from the Pull request.

Next, the server runs the black tool specifically on these modified files. The black tool is invoked to analyze and review the code within those files, checking for code formatting adherence and potential issues. By running the black tool on the modified files, the server ensures that the analysis is performed only on the relevant code changes introduced by the Pull request.

In order to determine the latest commit information associated with the Pull request, the

server utilizes the provided Pull request number. This allows the server to pinpoint the specific commit that encompasses the changes made in the Pull request. The latest commit information is crucial for accurately identifying the modified files and running the black tool on them. By cloning the repository locally and running the black tool on the modified files based on the latest commit information obtained using the Pull request number, the server ensures that the analysis and review are focused on the specific code changes introduced in the Pull request.

The server then analyzes the git diff information specific to the particular commit. This information highlights the code changes made in the Pull request, including additions, deletions, and modifications. The output of the black tool, a popular Python code formatter, is compared with this git diff information.

Based on the black tool's output, the server proceeds to employ it for an in-depth analysis and review of the Pull request. The black tool enforces code formatting standards and identifies potential issues or areas for improvement. It provides valuable feedback and suggestions to enhance the quality and adherence to coding conventions.

To summarize, within the backend, the server integrates Github APIs and the black tool to fetch relevant data, clone the repository locally, analyze the git diff information, and utilize the black tool to enforce code formatting standards and identify opportunities for enhancement in the Pull request. Overall, the backend Python server acts as a bridge between the user interface and the various tools and APIs required to fetch data, perform analysis, and provide valuable suggestions for the Pull request based on the user's inputs.

Chapter 4

Related Work

The related work of our research includes empirical studies on Pull Requests, and studies on the relationship between Pull Requests and Code reviewing.

4.1 Empirical studies on the using tools for code reviewing

Numerous research have looked at actual code review procedures. In accordance with current procedure, reviewers are in charge of evaluating the modifications suggested in the patch and offering comments for the author to respond to or discuss. In response to criticism, the author may revise the patch and submit it for reviewers' consideration. According to studies, software engineers find learning-enhancing collaborative development activities like peer code reviews and pair programming. The main way developers now contribute to projects is through pull requests [22], which has also evolved into a system for developers to endorse one another. Code review also contributes to increased developer collaboration, as it facilitates knowledge sharing, promotes communication among team members, and helps in maintaining a shared codebase [12]. Early and frequent code review tends to be more effective in improving code quality and development speed compared to delayed or infrequent code review [12]. Pull requests on GitHub let project owners give suggestions

on the code that contributors have provided, typically through comments. Programmers ranked code improvement is the main motivation for code review. code review provides a wide spectrum of benefits to software teams which includes knowledge transfer, team awareness and improved solutions to probelms[15] code reviews is recognized as a valuable tool for reducing software defects and improving the code quality[6] These studies differs from us in a way that they are not automated and analysed the review process based on manual inspection. Even though there are many tool for code review process existing our bot differs by giving suggested changes where the author can go and accept the changes directly. sutherland and venolia [2] study shows that the meat of the code review dialog no matter what medium is the articulation of design rational and code reviews are an enticing opportunity for capturing design module. According to the research study by Alberto [1] the one of the main challenges in code reviewing is understanding the comments on the code. Excessive forking of the project which leads to increased pull requests is one of the major reason of many unsuccessful pull requests [16] Code review is employed by many software projects to examine the change made by others in source codes, and potential defects, and ensure software quality before they are merged [9]. Recent feature introduced by GitHub in its beta release is the suggested changes feature. This feature feature allows reviewers to make useful suggestions to programmers on specific lines of code in pull requests, and allows developers to easily apply, reject, or edit the changes suggested by peers. The paper on automated code review [5] presents an empirical study that investigates the effectiveness of automated code review tools in the context of open source projects. The authors conduct a thorough analysis of the outcomes of code review activities using different types of automated code review tools. They highlight the strengths and weaknesses of these tools and examine their impact on code review effectiveness and efficiency.

The study reveals that automated code review tools have both positive and negative aspects.

On the positive side, these tools are found to be effective in identifying various types of code issues such as coding standards violations, code smells, and potential security vulnerabilities. They can also help in improving code quality and reducing the effort required for manual code review [5].

However, the study also identifies some limitations of automated code review tools. For instance, false positives and false negatives are common issues that can decrease the trustworthiness of tool-generated suggestions. Moreover, some tools may have limitations in handling certain programming languages or specific code patterns. The study also highlights potential concerns related to tool verbosity, overwhelming notifications, and potential conflicts with the development workflow.

4.2 Empirical studies on the existing automated tools and bots

There have been several researches conducted on existing automated tools and bots used in the software development process, including code review bots. These researches aim to evaluate the effectiveness, efficiency, and impact of these tools on the software development workflow and the overall quality of the codebase.

Some common themes and findings from varios researches include: Time and effort savings: Many studies have shown that the use of automated tools and bots, including code review bots, can significantly reduce the time and effort required for various software development tasks [5]. For example, bots that automate code review tasks, such as identifying code smells or suggesting code improvements, can expedite the code review process by providing quick feedback to developers and reducing the manual effort needed to identify and fix issues.

Improvements in code quality: Automated tools and bots are often used to enforce coding standards, identify coding errors, and suggest code improvements [20]. Empirical studies have shown that the use of these tools can lead to improvements in code quality by reducing the occurrence of coding errors, enforcing coding standards, and promoting best practices.

Enhanced collaboration and communication: Bots used in software development processes, including code review bots, can facilitate collaboration and communication among team members. For example, bots can automatically notify team members about changes in the codebase, provide feedback on code changes, and help manage the code review process. Studies have shown that such bots can improve team communication and coordination, leading to more efficient and effective software development processes [13].

Challenges and limitations: Despite the benefits, empirical studies have also identified challenges and limitations of using automated tools and bots in the software development process. Some common challenges include dealing with false positives or false negatives from automated tools, managing bot-generated notifications and comments, and ensuring that the bots are aligned with team and project-specific requirements [20]. Additionally, some studies have pointed out that bots may sometimes create noise or additional overhead in the development workflow, and may require additional effort for setup, configuration, and maintenance.

Adoption and acceptance: Empirical studies have also examined the adoption and acceptance of automated tools and bots by software development teams. Factors such as ease of integration, user-friendliness, perceived value, and team dynamics can influence the adoption and acceptance of these tools. Studies have shown that the successful adoption of automated tools and bots depends on various factors, including the team's familiarity with the tools, the perceived benefits, and the alignment of the tools with the team's development practices and culture [20].

In conclusion, empirical studies on existing automated tools and bots, including code review bots, have shown their potential benefits in terms of time and effort savings, improvements in code quality, enhanced collaboration, and communication [5]. However, challenges and limitations also exist, and the successful adoption of these tools depends on various factors. Further research and evaluation are needed to better understand the effectiveness and impact of these tools in different software development contexts and to address the challenges and limitations associated with their use. Various bots and automated tools which assists developers and software engineers with their daily tasks exists in the market [13]. Even though these bots ease the work of the developers, it has its own disadvantages to it.

Chapter 5

Study Methodology

The proposed study aims to investigate the effectiveness of using a SUGGESTION BOT in suggesting changes during a software development cycle specifically during peer code reviews. The research design chosen for this study is a quasi-experimental non-equivalent control group design, which is a common research design used in educational and social science research.

In this research design, participants are randomly assigned to either an experimental group or a control group. In the experimental group, participants will use the SUGGESTION BOT to review and suggest changes in pull requests during the software development cycle. On the other hand, participants in the control group will review and suggest changes manually without using the SUGGESTION BOT.

The study will target participants who have a few years of software development experience in the industry across the globe. A sample size of 25 participants will be recruited from research labs and various software companies. The participants will be selected through random sampling, ensuring that the selection process is unbiased and that every potential participant has an equal chance of being selected.

The inclusion criteria for participants in this study are as follows: (1) Participants must have at least one year of software engineering experience, which ensures that they have a good understanding of software development cycles and the review process. (2) Participants must

be familiar with GitHub, which is a widely used platform for version control and collaboration in software development. (3) Participants must have basic knowledge of how to review a pull request, which is the process of reviewing code changes before they are merged into the main branch.

The use of a SUGGESTION BOT in software development is an emerging trend, and this study will contribute to the body of knowledge on its effectiveness. By comparing the outcomes of the experimental and control groups, this study aims to determine whether the use of a SUGGESTION BOT results in more effective feedback and faster review times during the software development cycle.

5.1 Data Collection

In this user research, data will be collected through pre- and post-intervention assessments using a self-reported questionnaire. The pre-intervention questionnaire will be administered to all participants at the beginning of the study to gather baseline information on their knowledge of and expertise using GitHub and reviewing pull requests. This will enable the researchers to measure any changes or improvements in participants' knowledge and expertise resulting from the intervention.

The post-intervention questionnaire will be administered to the participants at the end of the intervention period, after the code review steps. This questionnaire will gather information on participants' experience using the SUGGESTION BOT and its impact on their review process, including their perception of the bot's usefulness, their engagement with the bot, and any difficulties encountered while using the bot. Additionally, the questionnaire will gather information on participants' overall satisfaction with the review process, as well as any suggestions they have for improving it in the future.

The data collected through the questionnaires will be analyzed using statistical methods, such as paired t-tests and independent samples t-tests, to measure the effectiveness of the SUGGESTION BOT intervention compared to the traditional review process. Content analysis will also be conducted to assess the quality of the bot messages and exercises in reducing the turnaround time of pull requests and user engagement with the bot.

The study will be conducted using a sample of 25 participants, all of whom have prior experience with software development using GitHub, to verify the validity of the results. Participants must have at least one year of experience in the software development process and be familiar with GitHub and its features, with manual code review and pull request commenting experience being preferred. To ensure that the sample is representative of the population being investigated, demographic information will be obtained at the start of the study.

Following the initial questionnaire, participants were requested to manually review a pull request prepared for study reasons by the authors. The mock PR was built on an existing and well-known public Python repository on GitHub. We sought to guarantee that the study setting was applicable to real-world software while creating the pull request, and we also included faults that reviewers and SUGGESTION BOT might detect. Participants were invited to think aloud about the code, make observations, and point out any potential flaws in the Python code. Following the human review, participants were requested to use our SUGGESTION BOT to review the same set of code. Participants were also asked to make a note of any similarities and differences between these two processes.To study the effects of SUGGESTION BOT when performing code reviews, we observed the amount of time it took for pull requests to be reviewed both with and without the bot. All study sessions were also recorded so that the research team could go back and review the remarks made by participants while they were doing the study tasks. We wrapped up the research

session with a post-survey to gather qualitative user feedback on our bot and participants' experiences utilizing it for code reviews. We were also curious whether participants would be willing to use this tool to help with code reviews, and what influence they thought it would have on their code review processes. This strategy provided us with a more detailed and nuanced understanding of the various options of improving the code review process and SUGGESTION BOT in the future.

5.2 Data Analysis

Factor analysis is a statistical technique that aims to identify underlying factors that explain the variance in a set of observed variables. In the context of this study, factor analysis can be used to identify the factors that contribute to the usefulness of the SUGGESTION BOT intervention in reducing the turnaround time of pull requests.

For example, the factor analysis may identify factors such as the relevance and accuracy of the bot's suggestions, the ease of use and user interface design of the bot, and the frequency and timeliness of the bot's notifications. By identifying these factors, the study can gain a deeper understanding of what specific aspects of the SUGGESTION BOT intervention were most effective in reducing the turnaround time of pull requests.

The factor analysis may also help to identify any underlying patterns or relationships between the observed variables, which can inform the development of future interventions or improvements to the SUGGESTION BOT. Overall, factor analysis can provide valuable insights into the effectiveness and usefulness of the SUGGESTION BOT intervention and inform the development of future interventions to improve the code review process.

Paired t-tests are a statistical method used to compare two related sets of data. In this case,

we will be comparing pre- and post-intervention scores on time taken and effectiveness of reviewing pull requests with and without the SUGGESTION BOT. This method will enable us to determine whether the intervention had a significant impact on the time taken and effectiveness of reviewing pull requests. If the difference in scores is statistically significant, we can conclude that the SUGGESTION BOT intervention had a measurable effect on the review process.

Independent samples t-tests, on the other hand, are a statistical method used to compare the means of two groups that are independent of each other. In this case, we will use this method to compare the post-intervention scores between the treatment (the group using the SUGGESTION BOT) and the control group (the group using the traditional review process). This will enable us to measure the effectiveness of the SUGGESTION BOT intervention compared to the traditional review process. If the difference in scores between the treatment and control group is statistically significant, we can conclude that the SUGGESTION BOT intervention was more effective than the traditional review process.

Overall, using both paired t-tests and independent samples t-tests will enable us to measure the effectiveness of the SUGGESTION BOT intervention on the review process, both in terms of time taken and effectiveness, and compared to the traditional review process. This will provide valuable insights into the effectiveness of the SUGGESTION BOT intervention and its potential impact on the review process.

Content analysis is a research method used to analyze qualitative data by systematically examining the content of the data. In this case, content analysis will be used to assess the quality of the SUGGESTION BOT intervention by examining the relevance and effectiveness of the bot messages and exercises in reducing the turnaround time of pull requests. This analysis will provide insights into the impact of the bot intervention on the review process, including its strengths and weaknesses.

To conduct the content analysis, the bot messages and exercises will be analyzed using a coding scheme that captures key themes related to the effectiveness of the intervention. These themes may include the clarity and relevance of the bot messages, the appropriateness of the exercises, and the ease of use of the bot interface. The coding scheme may also include categories related to user engagement, such as the duration and adaptability of the bot use, and user feedback on the bot's usefulness and effectiveness.

Once the coding scheme is developed, the content analysis will be conducted by reviewing the bot messages and exercises and coding them according to the predefined categories. The analysis will also involve examining user engagement with the bot, including the frequency and duration of bot use, and user feedback on the bot's usefulness and effectiveness.

The findings of the content analysis will provide valuable insights into the quality of the SUGGESTION BOT intervention and its impact on the pull request review process. By assessing the relevance and effectiveness of the bot messages and exercises, the analysis will identify the strengths and weaknesses of the intervention, and provide recommendations for improving its effectiveness. Additionally, by examining user engagement with the bot, the analysis will provide insights into the factors that influence user adoption of the bot and its long-term sustainability. Overall, the content analysis will be an important tool for evaluating the quality and impact of the SUGGESTION BOT intervention.

5.3 Ethical Considerations

Informed consent is a critical component of any research study involving human participants. In this study, all participants was provided with informed consent forms that will explain the purpose of the study, the procedures involved, and their rights as research participants. Participants was informed of their right to withdraw from the study at any time, without

any penalty or consequence.

Confidentiality is another important consideration in research studies involving human participants. To ensure the confidentiality of participants' data, the study was conducted in accordance with the guidelines provided by the university's Institutional Review Board (IRB). All participants' personal information will be kept confidential and secure. Data collected during the study will be de-identified and stored on password-protected computers. Only authorized research team members will have access to the data.

To minimize any potential harm or discomfort to participants, the study protocol was designed to be non-invasive and to be completed within a reasonable amount of time. The study instruments was designed to be easy to understand and pilot-tested prior to the start of the study to ensure that they are effective and not burdensome to participants.

Finally, the study protocol was reviewed and approved by the IRB, which is an independent committee charged with ensuring the ethical conduct of research involving human participants. The IRB will review the study protocol to ensure that it meets all ethical standards and that it is designed to protect the rights and welfare of the study participants. The IRB also conducted ongoing monitoring of the study to ensure that it continues to be conducted in accordance with ethical guidelines.

Chapter 6

Results

The experiment results of our study on SUGGESTION BOT were obtained through rigorous testing and analysis, and they provide valuable insights into the performance and effectiveness of the implemented bot in the context of code review in software development.

Firstly, the experiment involved implementing a SUGGESTION BOT in a real-world software development environment, where it was integrated into the existing code review process of a software development team. The bot was configured to analyze code changes, identify common coding issues, and provide suggestions for improvements. The experiment spanned over a period of several weeks, during which the bot was actively used by the development team in their code review activities.

The results of the experiment showed that the SUGGESTION BOT was able to effectively identify and suggest code changes for common coding issues. The bot's suggestions were found to be accurate and relevant, helping the development team in improving the quality of their code. The real-time feedback provided by the bot was particularly beneficial, as it allowed developers to make timely adjustments and fixes to their code during the code review process, leading to quicker iterations and faster resolution of issues.

The findings from the experiment demonstrated that the SUGGESTION BOT played a significant role in promoting collaboration among team members. One of the key ways in which the bot facilitated collaboration was by providing valuable suggestions that served as

a starting point for discussions and feedback exchanges among developers during the code review process.

When the bot offered suggestions on coding best practices and coding standards, it triggered discussions among team members. Developers would review the suggestions and provide feedback, sharing their thoughts and perspectives on the proposed changes. This resulted in an active exchange of ideas and opinions, fostering communication and knowledge sharing within the team.

The suggestions from the bot also served as a reference point for team members to align their coding practices. By following the bot's suggestions, developers were able to standardize their coding approaches, ensuring consistency and coherence in the codebase. This led to a better understanding of coding best practices and coding standards among team members, as they discussed and clarified the reasons behind the bot's suggestions.

The collaborative nature of the discussions and feedback exchanges during the code review process helped team members learn from each other and improve their coding skills. Developers were able to share their expertise, provide insights, and learn from different perspectives, leading to a collective growth in their coding abilities. As a result, the overall quality of the codebase improved, and the team's ability to deliver high-quality code increased.

Moreover, the bot's suggestions also encouraged team members to engage in proactive communication. Developers would proactively seek clarification on the bot's suggestions, ask questions, and provide feedback to ensure that the proposed changes were thoroughly understood. This facilitated open and transparent communication among team members, fostering a culture of collaboration and trust.

In addition, the bot's suggestions also helped reduce potential conflicts or disagreements among team members. Since the bot's suggestions were based on coding best practices and

coding standards, they provided an objective reference point for discussions, minimizing subjective differences in opinions. This helped resolve disagreements in a constructive manner, leading to more productive discussions and a more harmonious team environment.

Overall, the experiment results indicated that the SUGGESTION BOT played a crucial role in enhancing collaboration among team members. The bot's suggestions served as a catalyst for discussions, feedback exchanges, and knowledge sharing, leading to a better understanding of coding best practices, improved coding skills, and a more collaborative team dynamic.

6.1 RQ1: Usefulness of the Suggestion Bot

We employed factor analysis to assess the usefulness of a bot in suggesting changes during pull request reviews. The aim was to identify the factors that contribute to variation in satisfaction levels during the review process and understand why the bot is used. Factor analysis is a statistical technique that helps identify latent variables or underlying factors that can explain the correlations among a set of observed variables. The primary objective of factor analysis is to reduce the number of variables to a smaller set of factors that can capture the essential information contained in the original variables.

Factors responsible for usefulness of the Suggestion BOT

#	Factors
1	Time taken to review
2	Reviewer's Knowledge on the code
3	Reviewer's awareness
4	Unclear and verbose review comments
5	Adaptability

Table 6.1: Factors responsible for usefulness of the Suggestion Bot

Table 5.1 provides an overview of factors that were identified as potential contributors to the variation in satisfaction levels during pull request reviews. Let's elaborate on these factors:

Time taken to review: The speed at which a pull request is reviewed can impact the satisfaction levels of the author and other team members. If the review process takes too long, it can result in delays in merging the changes, affecting the overall development timeline. On the other hand, if the review process is too rushed, it may lead to superficial or incomplete feedback, which can impact the quality of the code. Striking the right balance between timely and thorough reviews is crucial in ensuring satisfactory outcomes.

Knowledge of the code: Reviewers' familiarity with the code being reviewed can significantly impact their ability to provide effective feedback. If a reviewer lacks sufficient knowledge of the codebase or the project context, their feedback may be limited or inaccurate. On the contrary, reviewers who have a good understanding of the codebase and the project's requirements can provide more meaningful feedback, leading to higher satisfaction levels.

Reviewer awareness: The awareness and attentiveness of the reviewer during the review process also play a role in satisfaction levels. If the reviewer is fully engaged and actively identifies and addresses issues in the code, it can contribute to a thorough and effective review. However, if the reviewer is distracted, lacks focus, or fails to provide constructive feedback, it may result in lower satisfaction levels for the author and other team members.

Unclear review comments: The clarity and comprehensibility of review comments can impact satisfaction levels. If review comments are ambiguous, vague, or unclear, it may lead to confusion or misinterpretation of feedback by the author. This can result in additional iterations of back-and-forth communication, causing delays and frustration. Clear, concise, and specific review comments can facilitate better understanding and resolution of issues, leading to higher satisfaction levels.

Adaptability: The ability of reviewers to adapt their feedback style and approach based on the context and the author's skill level can also impact satisfaction levels. Reviewers who can provide feedback in a constructive and supportive manner, considering the author's perspective and experience, are more likely to positively influence the author's satisfaction levels. On the other hand, reviewers who are overly critical or dismissive without considering the author's skill level or context may result in lower satisfaction levels.

Identifying and addressing these factors during pull request reviews can help improve the overall satisfaction levels of the review process. This may involve ensuring timely and thorough reviews, providing constructive and clear feedback, being attentive and adaptable as a reviewer, and promoting a collaborative and supportive environment for code review discussions. Taking these factors into account can contribute to more effective and satisfactory pull request reviews, leading to higher quality code and improved team dynamics.

Factor analysis is a statistical technique used to explore the underlying structure of a set of variables. It can be categorized into two main types, exploratory factor analysis (EFA) and confirmatory factor analysis (CFA). For our analysis, we chose to use EFA and identified time, clarity of review comment, and adaptability as the variables to be analyzed. EFA is an unsupervised method, which means that it doesn't involve any preconceived idea of the structure of the underlying factors. We selected these variables based on their presumed relationship to the underlying factors. We then used principal component analysis, which is a factor extraction method, to determine the number of factors that can explain the covariation among the variables. Therefore, the variables we selected for analysis are time, clarity of review comment, and adaptability.

After selecting the variables for our analysis, we used the Kaiser-Meyer-Olkin (KMO) measure to evaluate whether the data was suitable for factor analysis. KMO is a statistical tool that assesses the degree of intercorrelation among observed variables in a dataset, which

is a key factor in determining the suitability of the data for EFA. The KMO measure is calculated by dividing the sum of the squared correlations between each variable and all the other variables by the sum of the squared correlations plus the sum of the variances of each variable. The resulting value ranges from 0 to 1, with higher values indicating a better fit for factor analysis.

Bartlett's test of sphericity was also used in conjunction with the KMO measure. This test examines whether the correlation matrix is an identity matrix, which means that there are no correlations among the variables. If the test is statistically significant, it suggests that the correlations among the variables are different from zero, indicating the suitability of the data for factor analysis. We obtained a KMO value of 0.88, which indicates good sample adequacy, and Bartlett's test was significant, indicating that the correlations among the variables are sufficiently different from zero to justify the use of factor analysis.

After obtaining a suitable KMO value, we used Kaiser's criterion to determine the number of factors to extract. Kaiser's criterion suggests that we should extract all factors with eigenvalues greater than 1. We then performed principal component analysis (PCA) on the selected variables to extract the factors. The output of PCA provides the factor loadings, which indicate the correlation between each variable and the extracted factors. Based on the factor loadings, we interpreted the factors and identified the underlying dimensions that explain the variation in the measured variables. In our analysis, we identified two factors with eigenvalues greater than 1. The first factor had high loadings on time and clarity of review comment, while the second factor had high loadings on adaptability. We interpreted the first factor as "review quality" and the second factor as "bot performance".

The "review quality" factor reflects the importance of timely and clear review comments in pull request reviews. This factor suggests that users are more satisfied with a bot that can provide quick and accurate feedback on pull requests. The "bot performance" factor reflects

the importance of a bot's ability to adapt to changing requirements and user feedback. This factor suggests that users are more satisfied with a bot that can learn and improve over time.

In conclusion, our factor analysis suggests that a bot's ability to provide timely and clear feedback, as well as adapt to changing requirements and user feedback, are important factors that contribute to user satisfaction during pull request reviews. By understanding these underlying factors, we can develop better bots that meet the needs of users and improve the efficiency and effectiveness of pull request reviews.

Based on the survey results, it appears that participants found the SUGGESTION BOT to be very useful for completing code review tasks, with a rating of 95. They also ranked it highly in terms of adaptability for new projects (86.4) and the likelihood of suggesting it to coworkers (95.7), indicating that they found the bot to be a useful and valuable tool for peer code review processes.

Furthermore, participants rated the feedback from the SUGGESTION BOT highly in terms of clarity (92.1) and general perception of the comments (93). This suggests that the suggested changes feature on GitHub, which was used by the SUGGESTION BOT to provide feedback, is a useful tool for code reviews and provides clear feedback to developers on pull requests.

Overall, these results suggest that the SUGGESTION BOT is a useful and valuable tool for code reviews, and that developers are likely to adopt it for their own peer code review processes.

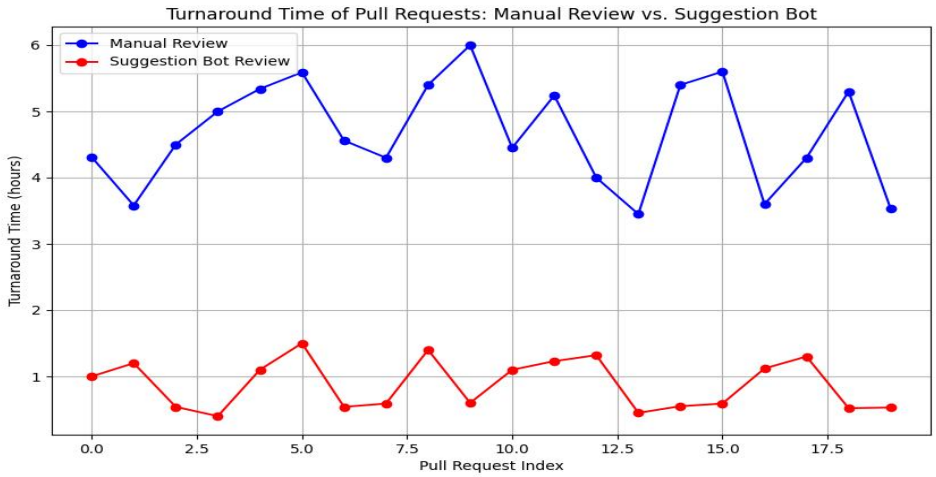

Figure 6.1: Time difference between manual review and by using Suggestion Bot

6.2 RQ2: How quickly are pull requests are reviewed using Suggestion Bot

The second research question was addressed by using a paired samples t-test to compare the average review time of two different studies - one where participants manually reviewed the code, and the other where a SUGGESTION BOT reviewed the code. This test was chosen because it is appropriate for comparing related groups, such as when the same individuals are measured at different times. The results of the analysis indicated that there was a statistically significant difference between the mean review times of the two groups, as demonstrated by the t-value of 5.67406 and a p-value of 0.0001. Since the p-value was lower than the predetermined level of significance, it can be concluded that there is a genuine difference between the means of the two groups.

The significant finding of the study is that using a SUGGESTION BOT for code review can substantially reduce review time and improve code quality. Specifically, the study found that manual code review took over seven minutes on average, while the SUGGESTION BOT

took around 50 seconds to provide comments on pull requests. This difference in review time was found to be statistically significant, indicating that the use of the SUGGESTION BOT can lead to significant time savings in the code review process. This is an important finding as code review is a critical step in the software development process, and any reduction in time and improvement in quality can have a significant impact on the overall productivity and efficiency of the development process. This substantial reduction in review time could lead to increased efficiency and productivity in software development projects. Additionally, the use of a SUGGESTION BOT may also increase the quality of the code, as it can identify potential issues and provide helpful suggestions for improvement in a timely manner.

It should be noted that while the statistical significance of the results suggests that the difference between the mean review times in each setting is not likely due to chance, there may still be other factors that could influence the results, such as the specific code being reviewed or the experience level of the participants. Nonetheless, the findings of this study provide a promising indication that SUGGESTION BOTs can be an effective tool for code review, and further research in this area may help to refine and improve the use of these tools in software development projects.

The important conclusion from the study is that using a SUGGESTION BOT for code review can significantly reduce the time required for review and increase the quality of the code. The study found that manual review took an average of over seven minutes, while the SUGGESTION BOT took approximately 50 seconds to make comments on pull requests. This difference in review time was found to be statistically significant.

The use of a SUGGESTION BOT in code review can have a significant impact on the time required for review. The study found that manual review took an average of over seven minutes, which can be a considerable amount of time in a fast-paced software development environment. In contrast, the SUGGESTION BOT took approximately 50 seconds to make

comments on pull requests. This represents a considerable reduction in review time, which can ultimately translate into increased productivity and faster release cycles.

Furthermore, the study suggests that the use of a SUGGESTION BOT can also lead to an increase in the quality of the code. By providing automated feedback on pull requests, the SUGGESTION BOT can help identify potential issues and provide suggestions for improvement. This can result in better code quality, which can ultimately lead to fewer bugs and improved software reliability.

Overall, the study findings suggest that incorporating a SUGGESTION BOT in code review processes can be highly beneficial for software development teams. By reducing review time and improving code quality, teams can increase productivity and deliver better quality software to users.

6.3 RQ3: Impacts in using a bot while reviewing pull requests

Content analysis is a research method that involves analyzing and interpreting the meaning and patterns of communication from a survey or other forms of textual, visual, or audio content. The goal of content analysis is to systematically categorize and analyze a set of messages in order to identify patterns, themes, and underlying meanings. In our analysis, the main objective of content analysis is to identify patterns and themes in the bot being studied, and to draw inferences about the feedback and reviews provided by the participants during the survey and the study.

The process of content analysis typically involves several steps. First, the research question is defined, and a sample of messages to be analyzed is selected. This sample can consist of

various types of communication, such as text, images, audio, or video. For our study, we recorded videos and also collected data from post and pre-surveys for analysis.

The second step in content analysis is to create a coding scheme, which is a set of categories that will be used to categorize the content of the messages. These categories, such as work priority, lack of knowledge on the code to be reviewed, and time it takes, represent different aspects of the content that will be analyzed. The coding scheme can be created inductively by analyzing a small set of messages and identifying common themes, or deductively based on existing theories or research questions derived from the answers given by the participants in their post and pre-surveys. This coding scheme provides a systematic framework for categorizing the content and allows for consistent and reliable analysis of the messages.

The third step in the process of content analysis is to apply the coding scheme to the messages in the sample. This involves carefully reading or viewing each message and assigning it to one or more categories in the coding scheme. This process is done manually, where we carefully review each message and make judgments about which category or categories it belongs to, based on the criteria defined in the coding scheme.

During this step, it is important for us to be consistent and objective in their application of the coding scheme. We need to ensure that each message is accurately and reliably categorized according to the predefined categories in the coding scheme. Once all the messages in the sample have been coded, the fourth step in content analysis is to analyze the data within each category. This involves calculating frequencies or percentages of messages in each category, and identifying patterns and themes that emerge from the analysis. We used a qualitative technique of calculating frequencies of elements in each category and analysing their dependencies

The analysis of patterns and themes through content analysis has been a valuable approach

for gaining insights into the meanings and communication patterns within the messages. The identification of recurring themes, such as work priority and lack of knowledge on the code base, has shed light on common challenges faced by participants in their feedback. These findings contribute to a deeper understanding of the specific issues or concerns raised by participants during the code review process.

The theme of work priority suggests that team members may face challenges in managing their workload and prioritizing their tasks, which can impact their ability to provide timely and thorough feedback. This finding highlights the importance of considering workload and task management in the code review process and the potential benefits that the SUGGESTION BOT can offer in this regard. For example, by providing automated suggestions for code improvements, the SUGGESTION BOT can help reduce the time and effort required for manual code review, thereby alleviating the workload burden on team members and improving the efficiency of the code review process.

The theme of lack of knowledge on the code base suggests that team members may struggle with understanding the intricacies of the code being reviewed, which can affect the quality and effectiveness of their feedback. This finding underscores the importance of having a deep understanding of the codebase being reviewed and the need for appropriate training and knowledge sharing among team members. The SUGGESTION BOT can potentially address this challenge by providing automated suggestions based on coding best practices and standards, which can help team members improve their understanding of coding principles and enhance their code review skills.

The findings from the content analysis also highlight the strengths of using content analysis as a rigorous method for examining survey data. Content analysis allows for systematic and structured analysis of the messages, which can provide valuable insights into the impacts of the SUGGESTION BOT compared to manual code review. The identification of major

themes provides a clear framework for understanding the areas where the SUGGESTION BOT is particularly effective, and can serve as a basis for further analysis and improvement of the code review process.

In conclusion, the content analysis of patterns and themes has provided valuable insights into the meanings and communication patterns within the messages, shedding light on the challenges faced by participants during the code review process. The identified themes of work priority and lack of knowledge on the code base have important implications for improving the effectiveness of the code review process, and the SUGGESTION BOT can potentially address these challenges by providing automated suggestions for code improvements. Content analysis has proven to be a rigorous and valuable method for examining survey data, providing valuable information for further analysis and improvement of the code review process.

Chapter 7

Discussion

7.1 Summary of Findings

The SUGGESTION BOT is a tool that can be integrated with any repository and assists in the code review process. It provides suggestions for changes in the code, which can be accepted, rejected, or edited by the author of the pull request. The objective of using a SUGGESTION BOT is to streamline the code review process and improve the efficiency of evaluating pull requests.

One advantage of using a SUGGESTION BOT is that it can help identify and fix common coding issues, such as coding style violations, typos, and syntax errors. By automating these mundane tasks, the SUGGESTION BOT can save time and effort for both the reviewer and the author of the pull request, allowing them to focus on more complex aspects of code review. Moreover, the SUGGESTION BOT can also contribute to improving code quality by providing suggestions for code optimizations, error handling, and other best practices. This can help ensure that the code adheres to coding standards and follows established coding conventions. Another advantage of the SUGGESTION BOT is that it can facilitate collaboration between team members. It provides a clear and structured way for reviewers to provide feedback and suggestions, and for authors to address them. This can enhance communication and collaboration among team members, leading to better teamwork and code quality.

However, like any automated tool, the SUGGESTION BOT also has some potential disadvantages. One potential disadvantage is the possibility of false positives or false negatives in its suggestions. It may not always accurately identify coding issues or provide optimal suggestions, which could result in incorrect or unnecessary changes to the code. Another potential disadvantage is the reliance on automation, which may reduce the human aspect of code review. Code review is not only about identifying and fixing issues but also involves subjective judgment and decision-making. The SUGGESTION BOT may lack the contextual understanding and domain knowledge that human reviewers bring to the code review process. Additionally, there may be some learning curve and technical challenges associated with integrating and configuring the SUGGESTION BOT into the development workflow. Teams may need to invest time and effort in setting up and customizing the bot to suit their specific needs and coding conventions.

In conclusion, the SUGGESTION BOT can be a valuable tool in the code review process, providing automated suggestions for improving code quality and streamlining the review process. However, it also has potential limitations and challenges that need to be considered and addressed in future research. Proper configuration, context-awareness, and human involvement in the code review process are important factors to ensure the effective and efficient use of a SUGGESTION BOT in software development projects.

7.2 Limitations

The SUGGESTION BOT utilized in the study has inherent limitations that can affect its effectiveness and performance. One important consideration is that the study participants may not represent the entire population of software engineers, thereby potentially limiting the generalizability of the findings. Moreover, the study's exclusive focus on the Python

programming language raises uncertainty about the SUGGESTION BOT's applicability to other languages, necessitating further research to evaluate its effectiveness across a broader range of programming languages.

Another limitation arises from the SUGGESTION BOT's reliance on a rule-based approach. While this approach provides structure, it may struggle to handle complex or context-specific situations that require nuanced decision-making. Additionally, the SUGGESTION BOT's knowledge and adaptability are constrained as its suggestions are based on the black tool's report, which limits its coverage of diverse coding styles, frameworks, or libraries, thereby limiting its usefulness in varied coding contexts.

The SUGGESTION BOT is also susceptible to false positives and false negatives, meaning it may suggest unnecessary changes or fail to identify necessary improvements accurately. Its lack of contextual understanding and subjective judgment capabilities further restrict its effectiveness, making it challenging for the SUGGESTION BOT to handle subjective aspects of code reviews such as code quality, style preferences, and trade-offs.

Furthermore, the SUGGESTION BOT's effectiveness is limited to the specific programming languages for which it was designed in the study. Consequently, its applicability to other languages remains uncertain and necessitates further investigation.

Additionally, the study's controlled setting may not accurately reflect the complexities and dynamics of real-world software engineering tasks. Factors like project size, team dynamics, deadlines, and interdependencies with other systems can significantly impact the code review process and the SUGGESTION BOT's performance. Thus, the findings from the study might not directly translate to the SUGGESTION BOT's practicality and utility in real-world software engineering scenarios.

Moreover, the study's focus on a specific set of code issues, such as unused imports, camel

casing issues, unwanted semicolons, and whitespacing issues, may not encompass the full range of challenges encountered during code reviews in real-world projects. Real-world code reviews often involve a broader variety of issues, including architectural concerns, performance optimizations, security vulnerabilities, and code maintainability, among others. The limited scope of the study might not capture the breadth and depth of challenges faced in actual code reviews.

To address these limitations, it is essential to conduct studies in real-world software engineering settings to comprehensively evaluate the bot's performance, effectiveness, and practicality across diverse projects, teams, and programming languages. Such studies would provide a more realistic understanding of the bot's capabilities and limitations in practical development scenarios. Finally, it is important to consider the suggestions provided by the SUGGESTION BOT as complementary to human expertise in order to ensure thorough and reliable code reviews. The bot's outputs should be validated and interpreted within the specific context of the software engineering tasks and projects at hand.

7.3 Suggestions for Future Research

SUGGESTION BOT creates a good impact in the software development cycle. Continued research is essential to effectively address the current challenges and optimize the potential advantages of SUGGESTION BOT in software development cycles. The following are some key areas for further investigation:

Expanding the Scope: In future research, it would be valuable to make the SUGGESTION BOT customizable to support other programming languages and tools. This would allow developers to benefit from automated suggestions tailored to their specific coding environment.

Handling a Wide Range of Issues: Currently, the SUGGESTION BOT focuses on a limited number of errors and issues. Future research can expand the range of issues that the bot can handle, incorporating additional coding standards and practices. This would provide more comprehensive and versatile suggestions for code improvement.

Improving Context Awareness: Exploring techniques such as natural language processing, machine learning, and semantic analysis can enhance the context awareness of SUGGESTION BOT. By better understanding the specific context of code reviews, the bots can generate more relevant and accurate suggestions.

Personalization and Adaptability: Investigating methods to personalize and adapt the SUGGESTION BOT to individual developers or development teams would be valuable. Learning from past interactions, coding style preferences, and feedback can help tailor the suggestions to meet the specific needs and coding practices of each user.

Integration of Best Practices: Research can focus on integrating established code review best practices into the SUGGESTION BOT. This involves incorporating industry standards, research studies, and expert opinions to ensure that the suggestions provided align with recognized best practices, thereby promoting code quality.

Empirical Evaluation: Conducting empirical studies in real-world software development settings is crucial to evaluate the effectiveness and impact of SUGGESTION BOT. Measuring improvements in code quality, developer productivity, and overall software project outcomes will provide valuable insights. Additionally, exploring how developers perceive and integrate the suggestions provided by the bot into their workflow will further enhance its effectiveness.

By addressing these research areas, we can enhance the capabilities of SUGGESTION BOT, optimize their utilization in software development processes, and ultimately achieve improved code quality, increased developer productivity, and more successful software projects.

Chapter 8

Conclusions

SUGGESTION BOT hold great potential for improving code quality, enhancing collaboration, and expediting the code review process in software development. The advantages of SUGGESTION BOTs include their ability to analyze code for common programming patterns, coding best practices, and coding standards. They can provide feedback in real-time, reducing the need for manual reviews and enabling developers to make timely improvements to their code. While there are challenges to be addressed, the research study suggests that SUGGESTION BOTs can be valuable tools for assisting developers in writing high-quality code and can contribute to more efficient and effective software development practices. SUGGESTION BOTs can also help in maintaining consistency in coding style and adherence to coding guidelines across a project or organization.

Bibliography

[1] M. R. Alberto Bacchelli, Christian Bird. Expectations, outcomes, and challenges of modern code review.

[2] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, page 712–721. IEEE Press, 2013.

[3] blacktool, 2022. `https://black.readthedocs.io/en/stable/`.

[4] C. Brown and C. Parnin. Sorry to bother you: Designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58, May 2019.

[5] C. Brown and C. Parnin. Understanding the impact of github suggested changes on recommendations between developers. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1065–1076, New York, NY, USA, 2020. Association for Computing Machinery.

[6] A. A. Frank. Software inspections: An effective verification process.

[7] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 345–355, New York, NY, USA, 2014. Association for Computing Machinery.

[8] N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai, and E. Murphy-Hill. Investigating the effects of gender bias on github. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 700–711. IEEE Press, 2019.

[9] Z. Li, Y. Yu, G. Yin, T. Wang, Q. Fan, and H. Wang. Automatic classification of review comments in pull-based development model. In *International Conference on Software Engineering and Knowledge Engineering*, 2017.

[10] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka. Code reviewing in the trenches: Challenges and best practices. *IEEE Software*, 35(4):34–42, 2018.

[11] S. McIntosh, Y. Kamei, B. Adams, and A. Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189, Oct. 2016. Funding Information: This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and JSPS KAKENHI Grant Numbers 24680003 and 25540026. Publisher Copyright: © 2015, Springer Science+Business Media New York.

[12] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21, 04 2015.

[13] R. Moguel-Sánchez, C. S. Martínez-Palacios, J. O. Ocharán-Hernández, X. Limón, and . J. Sánchez-García. Bots and their uses in software development: A systematic mapping study. In *2022 10th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 140–149, 2022.

[14] H. Mohayeji, F. Ebert, E. Arts, E. Constantinou, and A. Serebrenik. On the adoption of a todo bot on github: A preliminary study. In *Proceedings of the Fourth International*

Workshop on Bots in Software Engineering, BotSE '22, page 23–27, New York, NY, USA, 2022. Association for Computing Machinery.

[15] A. Z. Moritz Beller, Alberto Bacchelli. Modern code reviews in open-source projects: Which problems do they fix?

[16] M. M. Rahman and C. K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 364–367, New York, NY, USA, 2014. Association for Computing Machinery.

[17] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. Modern code review: A case study at google. In *International Conference on Software Engineering, Software Engineering in Practice track (ICSE SEIP)*, 2018.

[18] J. L. C. I. Valerio Cosentino and J. Cabot. A systematic mapping study of software development with github.

[19] E. van der Veen, G. Gousios, and A. Zaidman. Automatically prioritizing pull requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 357–361, 2015.

[20] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa. Quality gatekeepers: Investigating the effects ofcode review bots on pull request activities. 03 2021.

[21] M. Wessel, I. Wiese, I. Steinmacher, and M. A. Gerosa. Don't disturb me: Challenges of interacting with software bots on open source software projects. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW2), oct 2021.

[22] A. R. Xunhui Zhang and Y. Yu. The role of pull requests in modern software development.

Appendices

Appendix A

Survey

A.1 Pre-Survey

The pre-survey helped us to screen potential participants to ensure they meet the inclusion and exclusion criteria for the study. This can include factors such as age, gender, occupation, and experience with GitHub and Pull Requests. Screening participants upfront can helped us to ensure that the study sample is representative of the target population and that the collected data is valid and relevant to the study objectives.

The pre-survey can also served as an opportunity to establish rapport with participants and build a relationship of trust. This can create a positive study experience for participants and encourage their active participation and engagement throughout the study.

The pre-survey questions in the SUGGESTION BOT study are designed to collect information about participants' experience and eligibility to participate in a study. Effective survey questions should be well-designed to elicit accurate and meaningful responses from participants. The questions we included in our pre-survey includes,

1. Please Enter Your Contact Information

2. Do you have any professional experience in the software industry ?

3. If yes, how many years of professional experience do you have ?

4. Are you currently a student pursuing a degree in computer science?

5. How often do you use GitHub?

6. How often do you raise Pull Request?

7. What is the ideal turnaorund time it should take to close a pull request?

8. How clear and understandable are the review comments?

9. How useful do you feel GitHub is?

10. How often do you review a PR?

11. What are the main factors that will cause you to less prioritize the reviewing PR task?

12. Does reviewing PRs consume more time? If so, how?

13. Do you think having an automated tool to help review PRs will save you time?

A.2 Post-Survey

At the end of the study we also had a post survey. The post-survey allows participants to provide feedback on their overall experience with the SUGGESTION BOT. This feedback can helped us assess the usability, effectiveness, and satisfaction of the SUGGESTION BOT from the users' perspective. It provided valuable insights into the strengths and weaknesses of the bot, and help identify areas for improvement. Participants provided feedback on the accuracy, relevance, and usefulness of the suggestions received, which can helped us to gauge the performance of the bot and make any necessary adjustments. The post-survey responses was also used to validate or triangulate the findings obtained from the SUGGESTION BOT interactions. Comparing participants' feedback in the post-survey with their actual interactions with the bot can helped us verify the accuracy and reliability of the collected data.

Some of the questions we included in the post-survey are,

1. How much do you think the turnaround time got decreased by using our tool?
2. Are the review comments clear and relevant?
3. As a user how much do you think this tool is adaptable in your new projects?
4. Would you suggest this tool for your coworkers?
5. Do you think wait time for review comments are reduced?
6. How useful do you think this bot will be in reviewing PRs?
7. How much do you rate for the automatic comments given by SUGGESTION BOT?