THE RELATION BETWEEN SENTIMENT IN PROJECT DEVELOPMENT ARTIFACTS AND DEVELOPER PERFORMANCE

SYED FATIUL HUQ
Exam Roll: 707

A Thesis
Submitted to the Masters of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

MASTERS OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

THE RELATION BETWEEN SENTIMENT IN PROJECT DEVELOPMENT ARTIFACTS AND DEVELOPER PERFORMANCE

SYED FATIUL HUQ
Exam Roll: 707

Approved:

Signature *Date*

03/01/2021

Supervisor: Dr. Kazi Muheymin-Us-Sakib

To *my parents*
for the inspiration, the support, and everything in between

Abstract

Sentiment analysis in software engineering refers to the extraction of developer emotions from collaborative artifacts in order to study insightful patterns in software development processes. Developers collaborate with one another in projects using artifacts such as Commits — instance of code change by developers — and Issues and Pull Requests— discussions regarding new problems to the software and their solution. The text in these artifacts is observed to contain emotional values: negative, positive and neutral sentiment. These values are a direct insight into the mood of developers during their development activities. Previous studies have discovered how their mood influences important development factors such as developer productivity, inactivity, and more. These prove that a developer's work is affected by their emotions. However, whether the quality of the code, a performance metric significant in software development and maintenance, follows the same trend, has not yet been observed. On that account, this thesis utilizes the artifacts mentioned to analyze the correlation between developer sentiment and their performance.

In this study, developer performance, defined as the quality of code submitted by a developer, is determined by the presence of software bugs. Bugs are incorrect code that inadvertently causes unwanted behaviour in the software, damaging system functionalities and user experience. To deduce the instance of developers introducing bugs to the system, a concept named Fix-Inducing Changes (FICs) is used. FICs refer to Commits where a bug is initially written. This thesis analyzes

the sentiment of artifacts related to FICS to ascertain how developer emotions can influence their code quality, whether they write buggy code or not.

To correlate developer sentiment and their performance, this study analyzes three collaborative artifacts — Commits, Pull Requests and Issues. These artifacts are mined from Java repositories in GitHub. Each artifact contains associated contents for instance, messages in Commits, reviews in Pull Requests and comments in Issues. For each artifact, these contents are fetched and categorized accordingly. Next, FICs are detected from the Commit history of repositories. To do so, Commits that fix bugs through code removal are identified. The removed code is used to trace back to Commits where that code was last written, which are then labeled as FICs. The artifacts are classified as buggy or clean based on their proximity to FICs. The textual input from the artifacts are then used for sentiment analysis, utilizing sentiment detection tools specializing in the domain of software engineering, namely Senti4SD and SentiStrength-SE. Lastly, statistical analysis using Wilcoxon rank sum test and Chi-square test of independence is conducted to observe the intended correlation.

The findings show that Commits with negative and non-neutral messages tend to contain software bugs. FICs are 6% more non-neutral than regular Commits. Bug fixing contributions are also associated with a comparatively more negative sentiment, with p-value less than $8.8e^{-16}$ indicating statistical significance. Next, artifacts that are more concerned with the source code contain more technical and neutral text compared to discussions. Commits contain 84% neutral text whereas Comments in Issues and Pull Request contain around 56%. Lastly, Reviews and Comments from both development tasks and discussion are correlated with FICs, positive and non-neutral ones being more likely to result in buggy code. Ones posted before FICs contain on average 22.6% less neutral text compared to regular Commits. These findings indicate that developer sentiment in collaborative artifacts is correlated to developer performance.

Acknowledgments

I would like to express my sincerest gratitude to all who have extended their assistance and guidance in the completion of this thesis. It has been the help and support of my supervisor, faculty and thesis committee members, research team members, classmates and my family that have actualized the work I present here.

I begin my appreciation with my supervisor, Professor Dr. Kazi Muheymin-Us-Sakib. Over the span of my masters thesis and beyond, his guidance and inspiration have been constant. My motivation to conduct this study, to explore the many domains of knowledge, to critique my work, and to assess myself through it has been possible because of his continual support and earnest encouragement.

Next, my humble thanks goes to the faculty and thesis committee members of Institute of Information Technology, University of Dhaka for their honest feedback and valuable counsel.

I extend my gratefulness towards the Distributed Systems and Software Engineering research group, whose members have helped my progress with knowledgeable insights and constructive criticism. I am indebted to my classmates for their support in the smallest of hurdles and the company throughout. The contribution of my family in this work and my journey so far is beyond a note of thanks.

My final gratitude is towards the Ministry of Posts, Telecommunications and Information Technology, Government of the Peoples Republic of Bangladesh for granting me ICT Fellowship - No.: 56.00.0000.028.33.093.19-427, date: 20.11.2019.

List of Publications

1. Syed Fatiul Huq, Ali Zafar Sadiq and Kazi Sakib, "Understanding the Effect of Developer Sentiment on Fix-Inducing Changes: An Exploratory Study on GitHub Pull Requests," 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2019, pp. 514-521, doi: 10.1109/APSEC48747.2019.00075.

2. Syed Fatiul Huq, Ali Zafar Sadiq and Kazi Sakib, "Is Developer Sentiment Related to Software Bugs: An Exploratory Study on GitHub Commits," 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 2020, pp. 527-531, doi: 10.1109/SANER48275.2020.9054801.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Sentiment analysis — the process of extracting emotions from natural text [1] —- has garnered significant interest in the field of software engineering. This field of research combines sentiment analysis with software repository mining [2], which is the process of collecting and analyzing data from software project management systems. These systems contain software code, manage development tasks, enable code changes and reviews, and house developers and contributors who converse, collaborate and conduct development activities [3]. Previous studies, in this field, performed sentiment analysis in these systems to understand the patterns of developer sentiment and extract its relationships with other aspects of the development process. Their analysis found that developers convey emotional messages in the discussions [4] and contributions [5] they make. Additionally, sentiment is discovered to be correlated with properties of the project and task [5, 6], developer productivity [7, 8, 9, 10], team distribution [5, 10], and more. However, none of these studies have yet correlated developer sentiment with developer performance, in terms of the quality of their code — how correct or buggy a developer's code is. This research studies this correlation by analyzing developer sentiment in different stages of the development life cycle and relating it to Fix-Inducing Changes (FICs) as a metric for developer performance. FICs are simply defined as code changes

by developers that introduce bugs to the system [11]. Since bugs cause erroneous outcomes to the software [12] and consequent losses to product and people [13, 14], its introduction is used as a metric to judge how well a developer performs.

This introductory chapter first elaborates on the motivation of the research, which discusses the utility of sentiment analysis for mining opinions and the progress of studies that apply it to the domain of software engineering. Once the rationale for this research is established, the research questions are formulated to construct a guideline for the analyses necessary. This thesis answers three research questions to study the correlation between developer sentiment and FICs. Details of the resulting correlation and other findings are then described as the contribution and achievement of this research. Lastly, the organization of this thesis is provided as a direction for the readers.

1.1 Motivation

The utility of sentiment analysis originates in understanding emotional trends in a large corpus. With the influx of large amount natural text in online mediums, sentiment analysis has seen a rise in popularity and applications. These include:

- Summarizing emotional trends in social media [15]. Social media is a popular medium for people to share their thoughts on a variety of matters. Sentiment analysis helps condense the outlooks.
- Extracting opinions from news sites [16], online blogs [17] and more.
- Understanding product reception from review sites [18]. It has become a common tools for businesses to utilize for summarizing and understanding how their products are being received by their end users.
- Assessing employee behaviour in organizational settings . In recent times, emotional expressions in the workplace have been considered an important

metrics for evaluating the behaviours and performance of the workers [19]. The social dynamics in organizations play a significant role in how employees interact with each other and how that affects their work [20].

Software development processes differ from general organizational operations in the full adaptation and utilization of the online space [3]. Online version control systems, such as Git, have become a key aspect of almost every software projects [21]. These systems integrate social aspects in software development processes [22], containing artifacts for developers to contribute through and collaborate upon. The artifacts hold natural text from the developers, a necessity in extracting emotional value. GitHub is one of the most popular online version control and project management systems [23] that provide contributors with multiple collaborative artifacts: Commits [24], Issues [25] and Pull Requests [26]. These artifacts represent a contributor's interaction with the code and other contributors regarding the software.

1. **Commits**: A Commit is a contributor's documented modification to the source code. Each Commit is associated with a Commit message where the contributor describes the changes made.

2. **Issues**: Issues represent a new problem to be resolved or feature to be added. It contains comments by contributors discussing the matter at hand and the ways to solve/add it. These discussions ultimately shape the solution.

3. **Pull Requests**: Pull Requests are created by contributors once they want to merge their code to the base branch of the software. It exists as a gateway to the main code where the new code is reviewed for further modification. The final code, if satisfactory, is added to the system.

These three artifacts either represent a developer's programmatic contribution or contain discussions that motivated the contribution. The motivation can be

better understood through sentiment analysis, which quantifies the emotion of the artifacts and generates a tangible value to judge the artifacts with.

These artifacts have been used in previous studies to extract emotional patterns and correlations with development aspects.

1. GitHub Commits have been mined to observe how properties like day of Commit, project language and personnel diversity are related to sentiment. Guzman et al. [5] observed that Java projects have the most negative Commits, and geographically diverse teams tend to have more positive ones. Moreover, Commits made on Monday have the most negative emotion in the week. Sinha et al. [6], analyzing a different set of repositories, show that Commits made on Tuesday contain more negative emotion. These indicate that developers tend to be less positive at the start of the week. They also experiment to see how emotion is related to the number of changed files in a single Commit. They found a positive correlation between the two factors. Both of these studies mined the Commits, calculated their sentiment and analyzed to observe patterns in relation to environment and circumstances.

2. Issue comments from GitHub and Jira were analyzed to observe the effect of emotions in Issue resolution activities. Jurado et al. [4] mined GitHub issues to understand the applicability of sentiment analysis in this context. They extract affective emotions — *anger*, *joy*, *sadness*, *fear* etc. — from the Issue comments. They found that contributors leave latent emotion in the discussions in Issue comments, which can be used for detecting anomalies, for instance, a comment with a high *fear* value warns of potential dangerous code. Ortu et al. [7] analyzed comments from Jira Issues to observe the relation between contributor sentiment and Issue resolution time. They observed that the happier the developers are, expressing *joy* or *love*, the shorter the Issue resolution time is. The opposite is also evident: negative

emotions correlate to longer issue fixing time. Mäntylä et al. [8] also mined Jira Issues, with the motive to see a relation between sentiment and developer productivity or burnout. For their study, they adopted the VAD (Valence-Arousal-Dominance) model. Results of their analysis showed that Valence differs with Issue type (*feature request* vs *bug report*), Arousal increases with Issue priority (*minor* to *critical*) and quick Issue resolution is related to increased Valence. These three studies show how effective the sentiment in Issue comments -— discussions regarding a software task — is for understanding the effect of sentiment.

3. Outside of Commits and Issues, studies analyzed bug reports for sentiment. Garcia et al. [9] analyzed the Gentoo project's developer sentiment through mail archives and activities in bug reports from Bugzilla. They found a pattern of contributors leaving the project, permanently or temporarily, after showing extreme positive or negative emotions.

4. Other than public projects, private ones were analyzed by Guzman et al. [27]. They extracted the flow of developer sentiment on three projects using collaborative artifacts like Commit messages, bug reports, emails etc. The empirical results stated that the sentiment in the artifacts correctly represented real life developer behavior.

Although the studies observe and prove correlations between developer emotion and aspects like developer activity, task resolution, project properties, no relation has been established with developer performance. To understand whether emotions in the discussions actually affect a developer's performance, sentiment needs to be related to a metric that represents the quality of code. Fix-Inducing Changes (FICs) [11] — code that accidentally introduces bugs to the system, inducing its fix in the future —- are such a metric. FICs are used for discovering and analyzing the introduction of bugs to a software. These can be detected from

developer Commits [11, 28] that document the changes a developer makes to the system. Studies have analyzed FICs to derive useful information on coding such as how fixes themselves can create new bugs [29] and its relation with code coupling [30]. The introduction of bugs can be regarded as a metric for quantifying contributor performance since it is degradation to the quality of code. Therefore, observing whether sentiment in collaborative artifacts is related to FICs can yield an understanding of the effect of emotions on developer performance.

1.2 Research Questions

Section 1.1 provided the basis of this research. In summation, this thesis aims to find a correlation between developer sentiment in software development processes and developer performance in the form of Fix-Inducing Changes. The following three Research Questions (RQs) are derived accordingly.

1. *RQ1: Do sentiment in developer contribution indicate Fix-Inducing Changes?*

   Developer contribution is defined as a developer's modification to the existing code of a software. The reason for the modification can be to fix a bug, add or enhance a feature, refactor code or more. Regardless of which, these modifications are submitted via Commits [24]. When submitting a Commit, the author or developer must add a message describing the changes made. This Commit message can be used as the textual input for sentiment analysis.

   Sentiment in Commit messages has previously been analyzed [5, 6, 31] to find its relation with day of submission, contributor distribution, change size, build status and more. To answer this RQ, the correlation can be established with Fix-Inducing Changes (FICs). Since FICs themselves are Commits, the direct link can be established with the message of a Commit and whether that Commit is an FIC.

2. *RQ2: What is the relation between sentiment in development tasks and Fix-Inducing Changes?*

   Development tasks incorporate contributions from a developer and the associated reviews and comments that assess the changes made. This enables collaboration between multiple members of the project, namely the developer who submits the code and reviewers who evaluate those. In GitHub, Pull Requests [26] are used for this process. It contains both the contribution — in the form of Commits — and the reviews — in the form of Comments and Reviews.

   Sentiment in code reviews has been analyzed [10] to find the prevalence of neutral emotions and a correlation between sentiment and time to resolve a review. Instead of resolution time, this RQ's aim is to find how the reviews impact the code quality. For each Pull Request, its contents can be segregated based on resulting Commits. This segregation can group contents that lead to a single Commit. The average sentiment of the group can be analyzed while the resulting Commit is checked to see whether it is an FIC or not. Statistical analysis can be conducted to observe whether reviews leading to FICs contain differing sentiment than reviews leading to non-buggy Commits.

3. *RQ3: How does sentiment in contributor discussion lead to Fix-Inducing Changes?*

   Contributors of a software project communicate with each other on topics related to software tasks, for example, bugs to be fixed or features to be added. In GitHub, these discussions occur on Issues [25]. The Comment feature in Issues are used by contributors for posting opinions, suggestions, queries regarding the task to be conducted. The discussions shape the way solutions are derived by developers. Hence, the code of a developer is influ-

enced by the discussions that precede it.

The effect of sentiment in developer discussion has been analyzed [4, 7, 8] and it is seen that sentiment influences the time taken resolve Issues. While these studies measure productivity to observe the impact of contributor sentiment, this RQ aims to correlate it with FICs to check its impact on developer performance. Issues that lead to an FIC and those that do not can be differentiated and their intrinsic sentiment calculated. The statistical difference can indicate whether a correlation exists between sentiment in contributor discussion and buggy code.

1.3 Contribution and Achievement

This research explores the different stages of software development lifecycle — developer contribution, development tasks and contributor discussions — to analyze the sentiment expressed and relate it to buggy code or Fix-Inducing Changes. The following are the outcomes of this research work.

1. **Bug-related code contribution contains more negative and non-neutral sentiment compared to clean code changes.**

   The sentiment in developer contribution are calculated from the messages in Commits. The Commits are categorized into four parts to indicate bug-related changes: Commits that add bugs, Commits that precede buggy code, Commits that fix bugs and Commits that both fix and add bugs. The sentiment in bug-related Commit messages are statistically differentiated from clean Commits — Commits that are not related to bug introduction.

   Bug-related Commit messages are found to be significantly more negative than their counterpart. It is inferred that a negative mood is not only associated with incorrect changes but also with bug fixing activities, which

is similar to the findings of [8]. Furthermore, in terms of subjectivity, bug-related Commits are also less neutral than clean Commits. This is inferred as developers being more opinionated in their messages and less formal or technical when working with or adding bugs.

2. **The closer a collaborative artifact is to the source code the more neutral its text.**

   The three RQs analyze text from three distinct stages of the development ecology. In terms of artifacts, the analysis is conducted on Commits, Pull Requests and Issues. The three artifacts have different proximity to the source code. Commits are the direct modification to code, and therefore the closest. Issues are discussion conducted on bug reports or feature requests, and hence further from the code. Pull Requests contain both discussions and a list of contribution to the source code, which places it in between the previous two. Before correlating sentiment to FICs, the average sentiment is detected from these artifacts collectively.

   It is seen that, in terms of subjectivity, Commit messages are the least neutral. The ratio of neutral text in Commit messages is approximately 84%. On the other hand, when taking the Comments in Issues and Pull Requests, the ratio of neutrality comes down to 56%. When differentiating between the average sentiment of Issues and Pull Requests, neutrality is less in Issues with the difference being statistically significant as measured by the Wilcoxon rank sum test. This shows how neutrality decreases, or emotions increase, as the artifact is situated further away from the source code. This is conforming to the fact that Commit messages are designated for technical description of changes made, with little scope for opinionated text. On the other hand, both Issue discussion and Pull Request reviews require the participants to debate on the best tactic to resolve a problem. This creates

scope for opinionated conversation in the Comments and Reviews.

3. **Positive and non-neutral discussions lead to buggy changes.**

   To understand how the collaboration among developers affect the introduction of bugs, sentiment analysis is conducted in two separate stages. First, review-oriented discussions from Pull Requests are analyzed. Here, for every Commit, the Reviews and Comments that preceded it are grouped and their sentiment scores averaged. This enables exact observation of discussion sentiment that lead to buggy code and those that do not. In the second stage, Issue Comments are taken for sentiment analysis. For correlation, instead of individual Commits, it is checked whether a whole Issue is linked to an FIC or to a Pull Request containing an FIC.

   In both cases, it is seen that the discussions that contain more positive and non-neutral text are more likely to end up in buggy code. While this is different from the negative Commit messages being FICs, both correspond to the lack of neutrality in the expression. This can be interpreted as a lack of technical and formal discussion when dealing with software modification leading up to an unwanted incorrect code.

1.4 Organization of the Thesis

Following are chapter-wise overviews that work as a guideline for readers:

Chapter 2: Background Study In this chapter, concepts regarding the study are elaborated. Three distinct fields are combined in this study: sentiment analysis, software repository mining and Fix-Inducing Changes (FICs). First the metrices, detection processes and challenges of sentiment analysis are described. Next, types of repositories used in repository mining, their artifacts and processes are reported. Lastly, software bugs and bug-related activities which include FICs are presented in detail.

Chapter 3: Literature Review on Sentiment Analysis in Software Engineering This chapter lists the previous studies that mined software repositories and conducted sentiment analysis. The studies are grouped based on the artifacts utilized: compound artifacts, Commits, Issues and code reviews. Elaboration of each study is provided, which include their analysis goals, use of collaborative artifacts, sentiment analysis processes and the resulting patterns and correlations.

Chapter 4: Sentiment in Developer Contribution and FIC In this chapter, the first Research Question (RQ) is answered. Commits from GitHub are mined, categorized based on context to software bugs, analyzed for sentiment and differentiated based on the existence of FICs. The chapter ends with the findings and threats to validity.

Chapter 5: Sentiment in Development Tasks and FIC RQ2 is answered in this chapter, by exploring the sentiment expressed in the contents of Pull Requests. The study design consisting of content categorization, sentiment analysis and FIC detection is described. This is followed by the results from the statistical analysis, along with the threats to validity.

Chapter 6: Sentiment in Contributor Discussion and FIC In the last contributing chapter, RQ3 is answered. This chapter analyzes Issue comments to understand the effect of contributor discussion on FICs. First the exploratory methodology describing GitHub content graph creation, dataset description and sentiment analysis is provided. The results include findings of the correlation, followed by the threats to validity.

Chapter 7: Conclusion This concluding chapter summarizes the methods adopted and contributions made in the preceding chapters, answering all three RQs. Lastly, the potentials of this thesis and the research domain in the future are listed.

Chapter 2

Background Study

With the emergence of text via online mediums, sentiment analysis has become a popular tool for easily and effectively understanding the opinions of the masses. The application domain of sentiment analysis has expanded from social media trends [15] and product review summaries [18] to the professional sphere [19]. Employee interactions in the scope of organizational activities has been analyzed to assess workplace satisfaction and performance [20]. Applications and studies of similar purview has extended upon the software engineering domain [32]. With the utilization of software repository mining, interactions among contributors of a software project can be extracted and analyzed for emotions [2]. Collaborative artifacts [33], available as tools of software projects, bind conversations with contributions, enabling correlation between the two. This study utilizes this connectivity to correlate the sentiment in the conversations with the qualitative output of contributions in the form of buggy code or Fix-Inducing Changes (FIC) [11]. This chapter contains the background information encapsulating all necessary components that construct this study: sentiment analysis, software repository mining and Fix-Inducing Changes.

2.1 Sentiment Analysis

Sentiment analysis is the study or process of extracting emotion from written text [1]. Also referred to as opinion mining, sentiment analysis is a branch of Natural Language Processing (NLP) [34] that focuses on human emotions in natural language. In this study, the terms "sentiment", "emotion" and "opinion" is used interchangeably because, despite their ambiguous distinctions, the terms convey a synonymous concept: the attitude of one or more individuals towards a topic [35].

It is an important tool for data mining efforts [36] with widespread domains of applicability:

- **Opinion summarization**: With the influx of textual data on the internet, sentiment analysis has seen a rise in popularity in analyzing the opinions of netizens [15]. Sentiment analysis has been applied in social media platforms to extract mass opinion on current events [37], political views [38] and more. Analysis of subjectivity has been conducted on personal blogs [17] and news site [16], to check both informal and official point of views.

- **Product review**: Another popular application of sentiment analysis lies in product reviews [18]. Businesses nowadays have enabled online forums and reviewing sites to gather feedback from their consumers. To summarize the numerous feedbacks posted, sentiment analysis is used [39]. Products include merchandise [40], movie [41] and other media, mobile applications [42] and more. The results of these analysis helps businesses assess their products, comparatively evaluate the features provided, fine tune marketing strategies and plan for future releases.

- **Workplace**: Other than public domains, sentiment analysis have also seen use in the professional sphere. Sentiment analysis has been applied on employee interactions via emails to assess their satisfaction and vitality [19]. It

has been observed that emotions expressed in communications among employees are influenced by organizational structures and job roles [20]. Further correlations have been made with after-hour work, geographic disparity, interpersonal communications and more.

- **Software engineering**: A recently popular trend has been the application of sentiment analysis in the domain of software engineering. Researchers in this domain have utilized sentiment analysis in multiple aspects, for instance, issue resolution time prediction [7, 8], developer turnover prediction [9] and more. Details on these applications are provided in Chapter 3.

Based on its broad applicability and rising potential, sentiment analysis has seen an emergence in research. Metrices for calculating sentiment values, along with the processes of calculation, have been explored widely in literature.

2.1.1 Sentiment Analysis Metrices

To quantify the emotions present in natural text, various metrices are proposed and applied. The metrices differ in their scope of identification and purpose of representation. Three prominent metrices in the literature of sentiment analysis in software engineering are described as follows.

Sentiment Polarity

Sentiment polarity refers to the separation of sentiment values based on positive and negative emotions. Sentiment analysis based on polarity is the most common metric in both the scope of this study and the broad domain. It is a simplistic approach but effective in distinguishing sentiment for data analysis.

This approach provides two information on the sentiment of a text:

1. **Polarity**: Whether a text conveys a positive message or a negative one, or whether no emotion is conveyed (neutral emotion).

2. **Strength**: How strong is the emotion conveyed in the text, regardless of which end of the spectrum it is.

The range of polarity values can be -1, 0 and +1 with the three values representing negative, neutral and positive emotions respectively. To incorporate the strength of the emotion, a range of -x to +x can also be used, where $|x| > 1$. The larger the value, the stronger the corresponding emotion. The existence of a zero value helps incorporate subjectivity — whether a text contains emotion or not — to the sentiment polarity approach.

Affective Emotions

Affective emotions refer to a specific emotional state among the many facets of the human emotion [43]. Sentiment polarity only calculates emotions on a single spectrum, not considering its complexities and varieties. Instead of single-range polar values, affective emotions include multiple types of emotions. Parrott's framework is one such classification of affective emotions [44]. The framework classifies emotions into a three-leveled tree structure, with each level expanding in granularity that more concretely defines the emotion. The primary level contains the following emotions:

- Love,
- Sadness,
- Anger,
- Joy,
- Surprise, and
- Fear

To exemplify the tree structure, *Sadness* is divided into six more specific emotions: *Suffering*, *Sadness*, *Disappointment*, *Shame*, *Neglect* and *Sympathy*. *Suffering* is further divided into *Agony*, *Anguish* and *Hurt*.

Affective emotions are used for better understanding and distinguishing the sentiment present in the text corpus [32, 7]. However, for mining purpose, and especially in the software engineering domain, it is not frequently utilized.

VAD

VAD refers to an emotional framework consisting of three dimensions: *Valence*, *Arousal* and *Dominance*. Unlike affective emotions, which are classified as part of a discrete approach, VAD represents a dimensional approach [45]. In the discrete approach, the emotional states are considered unique [43]. The actor is expressing either *Sadness* or *Joy*, but not both at the same time. In the dimensional approach, each state can exist simultaneously with differing strengths. The three major states or dimensions of VAD are as follows:

1. **Valence**: Valence refers to a subject's nature of attraction towards a topic [46, 47, 48]. In other words, it is similar to sentiment polarity as it demonstrates how positively or negatively the subject reacts to a topic in question.

2. **Arousal**: Arousal refers to the level or intensity with which an emotion is conveyed [46]. In terms of polarity, Arousal demonstrates how strongly the emotion is felt. If a subject shows a high Valence with high Arousal, it means that they have a strong positive response in the case.

3. **Dominance**: Dominance refers to the feeling of control over the emotion presented [49]. It is the subject's confidence in their emotions towards a scenario or topic.

The VAD metric is useful for analyzing sentiment for a variety of moods and expressions. While the previous two metrices provide distinct measures of certain

emotional states, VAD scores can combine the states according to the multiple dimensions and express complex forms like burnout or productivity [50].

2.1.2 Sentiment Analysis Methods

The methods of extracting sentiment from natural text have been explored in previous studies as an extension of NLP. Regardless of the metrices used, there are two principle methods used for sentiment analysis [51]. One process utilizes lexicons with predetermined emotional values, and the other uses machine learning methods understand the emotions latent in a text corpus. These are described in the following subsections.

Lexicon-based Classification

The lexicon-based classification method calculates emotional values using opinion lexicon. Opinion lexicons are constructed from words, phrases and idioms that express emotions. By predetermining lexicons with their common emotional values, sentiment can be determined of newer text corpus that contain those lexicons. Constructing the corpus of opinion lexicons can be done manually, but are usually associated with automated approaches to reduce time and effort consumption. There are two main automated approaches:

- **Dictionary-based approach**: Existing dictionaries consisting of all the words of a language are used in this approach [52, 53]. Instead of manually annotating all the words, it is iteratively conducted. The manual part is restricted to the first iteration. In the first iteration, a set of words is selected that have common emotional connotations and are assigned sentiment values accordingly. In the next step, words that are synonymous to those are given a similar value. For instance, if *good* is positive on first iteration, *well*, a synonym of *good*, is assigned a positive value on the second iteration. This process is repeated until all the words are covered.

While this approach is extensive, a downside lies in its lack of contextual annotation. Since the labeling of sentiment values is mostly dependent on the similarity of meaning in dictionaries, words that convey different emotions in different contexts are not given the correct value. For instance, in the context of movie reviews, the word *tragedy* is neutral, as it represents a genre. However, since it is a synonymous to *sad*, which is a negative word, *tragic* will also be labeled as negative.

- **Corpus-based approach**: Instead of using the common dictionary, the corpus to build the opinion lexicons is context-specific text. As in, to determine sentiment of movie reviews, the opinion lexicons are calculated from sample movie reviews. This approach is similarly iterative as the previous one, but the iterations are not restricted to synonyms.

  Relations between words are constructed using statistical or semantic methods. In the statistical approach, it is assumed that words with similar sentiment are stacked together [54, 55]. A word occurring more frequently with positive ones is labeled positive as well. The similar is true for negative words. If the frequency is equal for both polarities, it is regarded as a neutral word. In the semantic approach, the relationship is based on various semantic principles [56]. A hybrid approach combining the two is also adopted in other applications [57].

  This approach mitigates the contextual drawback imposed in dictionary-based approach. This is because corpus-based approach utilizes context-specific corpus to build its opinion lexicon.

Machine Learning

Machine learning algorithms are popular methods for sentiment analysis that do not depend on semi-automated annotation of opinion lexicons. Supervised learning

is commonly used in this domain due to the labeling of common sentimental texts. The algorithms range from probabilistic classifiers like Naive Bayes [58], Bayesian Network [59] and Maximum Entropy Classifier [60] to linear classifiers like Support Vector Machines [60] and Neural Network Classifiers [61].

Machine learning-based methods outperform lexicon-based classification at sentence-level contextualizing. Lexicon-based classification depend solely on the lexicon for calculating overall sentiment of a sentence or the whole corpus. However, sentences can be complex and an abundance of words showing one sentiment polarity does not necessarily mean an overall sentiment of the same polarity. An example can be the sentence: "The boy claimed to be an earnest [*positive*], hard-working [*positive*] and punctual [*positive*] student, but proved himself to be a liar [*negative*]." While the positive words outnumber negative ones, the sentence overall conveys a negative message. Simply summing up sentiment values from lexicons would yield an incorrect result. In this case, machine learning algorithms can train the model to detect contradictions in complex sentences and calculate sentiments accordingly.

2.1.3 Challenges of Sentiment Analysis

The process of sentiment analysis contains a variety of challenges that range from the accuracy of sentiment detection to the correctness of its applications. Two of its most prominent challenges present in the domain of software engineering are mentioned as follows.

- **Ambiguity of Natural Language**: Human to human communications are complex and its nuances are often undetectable in NLP [34]. For instance, ironic commentary or sarcasm cannot be detected in automated calculation of sentiment, labeling these incorrectly as a result. These issues are present in the scope of software engineering as well.

- **Contextual Dependency**: Sentiment differs greatly based on the context of the text corpus. Words and phrases expressing positive messages in product review sites may, for instance, convey negative emotion in news reports. Studies have also shown how models trained in one context do not perform accordingly in another [62]. In the domain of software engineering, contextual dependency is very prominent, with its abundance of technical terms.

2.2 Software Repository Mining

This study applies sentiment analysis on artifacts contained in software project management systems with the help of repository mining. Repository mining is a subset of data mining approaches that specifically works with the contents of software development processes [2]. This study mines repositories to extract textual and code-specific information that encapsulates developer contributions, development tasks and contributor discussions. This section elaborates on the various concepts that constitute software repository mining.

2.2.1 Types of Repositories

To integrate multiple developers working on a single software and to ease the management of both the code and the processes that alter it, software projects are stored online in repositories [3]. Over the years, a variety of repositories have been adopted by private and public software projects. These can be categorized in the following three types.

Version Control Systems

The fundamental component of a software is its source code, and its management is the primary aspect of software development processes. The broader the software project the larger its code base, requiring extra effort to better maintain it [63].

One of these maintenance efforts is version controlling the source code. Version control refers to storing the software code as a series of iterations [64], saving each change of the software. These iterations are commonly labeled as revisions.

Popular Version Control Systems include Git [65], Apache Subversion (also known as SVN) [66], Concurrent Versions System (CVS) [67], Mercurial [68] and more. Among these, Git is currently the most popular Version Control System with 72% of software projects adopiting Git [21]. Therefore, this study incorporates Git as a component of repository mining.

Git refers to revisions as *Commits*, which are submitted by multiple developers on a single code base. Each developer's version of the code is stored as a separate repository of itself, namely *branches*. As a Distributed Version Control System (DVCS) [69], Git provides faster operations and security against data loss. For mining purposes, however, it provides two very important features:

1. Git archives all the Commits of a software, tracing back to its origin. A long-term history of all versions of all files of a software system helps in code forensics and other maintenance activities [70]. It enables traceability, returning to previous versions to detect root cause analysis for concerns like bugs[11, 28, 29].

2. Each Commit not only contains information of the code changes but also associated data like, the developer's identity, a message describing the changes made and a timestamp of submission. These information combined provide important resources for analysis in repository mining.

Bug/Issue Repositories

Other than the source code, software projects require managing their tasks and contributors. Tasks can range from fixing bugs and adding new features to refactoring code and modifying documentations. Multiple tasks are completed concur-

rently by assigning each task to separate contributors. To manage these tasks, their assignments, prioritization and tracking their status in a centralized system that is accessible to all contributors, bug/issue repositories are created.

Popular bug/issue repositories include Jira [71], Bugzilla [72] and more. The primary features of these repositories include:

1. Bug/issue posting: End users and contributors alike can submit a post describing a bug, a request for new or upgraded features, and more. These posts inform the stakeholders of the software on the problems the system is facing. Additionally, they obtain recommendations on how to enhance the product to meet the evolving consumer needs.

2. Issue management: Once an issue is posted, the contributors assess the concern described. They can discard the issue if it is invalid or a duplicate. If not, they discuss about the concern among themselves and the submitter. These discussions lead to a better understanding of the issue, based on which the issue is prioritized among the several ones posted. Based on its priority and available developers, the issue is assigned with a specified deadline.

3. Issue tracking: After assigning an issue, its progress can also be tracked in these systems. Based on the progress by the assigned developer, other contributors can review the work. In the end, once the code satisfies a resolution, the issue is considered completed.

Hybrid Systems/Project Repositories

Hybrid systems combine the code management of Version Control Systems and task management of bug/issue repositories. These systems house the resources offered by both systems along with the processes that encapsulate those. While the code is maintained by VCS software, it is connected to the collaborative artifacts that manage issues.

For repository mining, these systems provide the most resources to work with. Not only do these provide artifacts and contents of both of the previous systems, but interconnecting these previously separated artifacts enable additional features to mine. For instance, Commits can be traced to their originating issues, contributor activities can be tracked for both development and collaborative activities, issue resolution activities can be directly mapped to the participating contributor, the assigned developer and the resulting code, and so forth.

Several project repositories populate the online space including GitHub [33], GitLab [73], BitBucket [74] and more. These repositories house numerous projects, ranging both enterprise and open source software. Among these, GitHub is the most popular system with over 40 million developers and 44 million repositories as of 2019 [23]. In this study, GitHub is used for mining projects and analyzing their artifacts.

2.2.2 Collaborative Artifacts

GitHub offers multiple types of features that range from administrative and managerial activities to testing and operation tools. This study focuses on GitHub's collaborative artifacts, which enable inter-contributor collaboration during the many stages of the development process, as detailed in Subsection 2.2.3. There are three principle collaborative artifacts in GitHub:

Commit

As elaborated in Section 2.2.1, Commits are the key components of Version Control Systems, which GitHub adopts. In GitHub, Commits are the central unit of change to a repository [24]. Each repository, fork and branch are made up of Commits submitted by one or more developers.

Figure 2.1 demonstrates a sample snippet of a Commit from the Mockito project [75]. It contains all the core components of a Commit: the Commit mes-

sage, information on the developer, timestamp of submission, a SHA to uniquely identify the Commit, and information on the changes made.

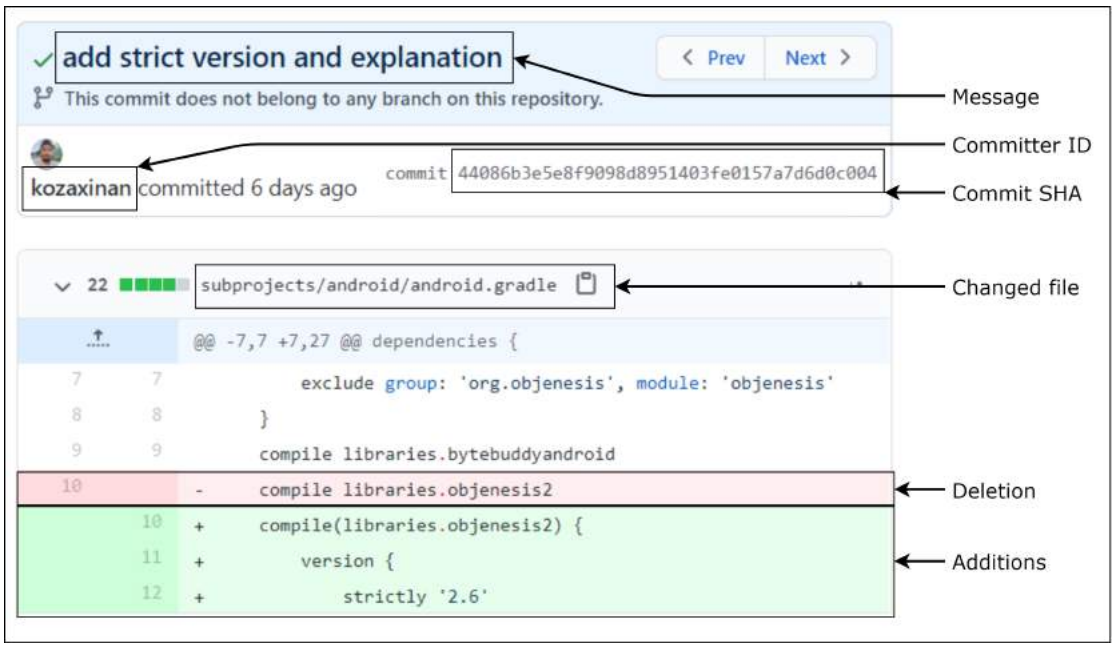

Figure 2.1: A sample Commit from Mockito

Due to GitHub's hybrid system, these information can be mined with added utilization. For instance, the developer ID is linked to the developer's other activities, which include comments on Issues or more Commits. The message can contain a link referencing an Issue that it resolves [76], connecting the two artifacts. Commits in GitHub are also listed in Pull Requests, associated with reviews and encapsulating development tasks. These aspects are utilized in this study.

Issue

Issues [25] are artifacts in GitHub adopted from its bug/issue repository roots. Issues are posts in GitHub that can comprise of multiple concerns or intents [77], including but not limited to:

1. Bug report: notifying the stakeholders of the software on a potential bug in the system, associated with information regarding its behaviour and source.

2. Feature request: a demand or suggestion for a new feature to the system directly from a consumer or based on their reviews.

3. Enhancement: a demand or suggestion for improving existing features.

4. Help post: a request for clarifying certain use cases of the software product from a user.

5. Documentation: a task concerning modifying the documentation of the software project to provide better insight into the system.

6. Community feedback: reviews on the software or processes of an open source project from the contributing community.

In Figure 2.2, a snippet of an Issue from the Mockito project [75] is displayed. It shows the primary components of GitHub Issues: a title summarizing the concern, a body expanding upon it, an ID for identification and referencing, ID of the user posting the Issue and a timestamp. Outside of these, GitHub also offers other contents with Issues, for instance, labels to better clarify the intent of the issue, milestones to define a product release before which this Issue needs to be resolved, assignees to show the contributors to whom this Issue is assigned to and more. Furthermore, Issues provide the features of a forum, where contributors and users can discuss about the Issue in the comments. All of these textual data can be utilized for sentiment analysis, as demonstrated in this study.

Pull Request

Pull Requests are collaborative artifacts that represent the conceptual fusion of Version Control Systems and bug/issue repositories [26]. In Pull Requests, concerns posted in Issues are resolved through a series of Commits from the assigned developer. The Commits are associated with comments and reviews from other contributors and designated reviewers. These discussions shape the final code,

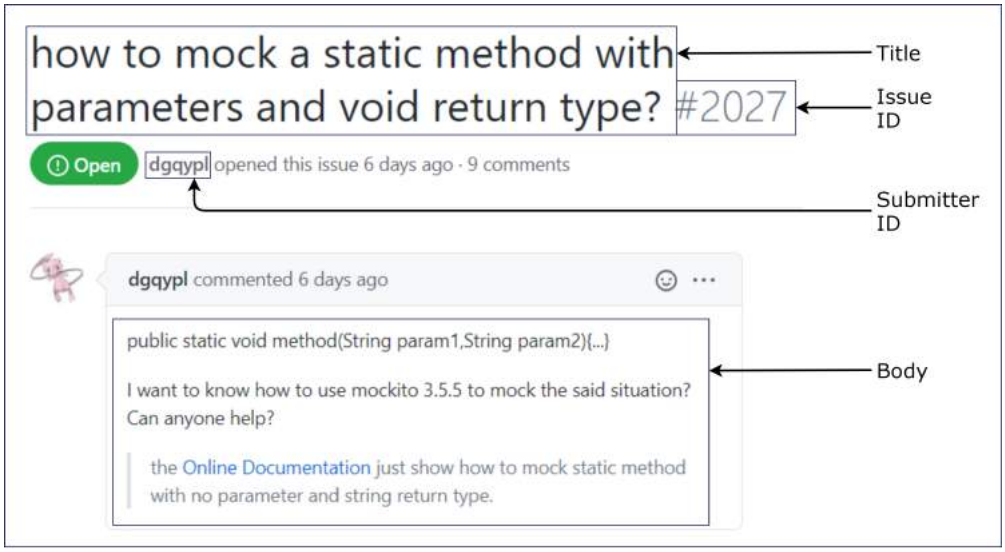

Figure 2.2: A sample Issue from Mockito

which becomes the resolution of the Issue. Once resolved, the code is merged with the base repository. Not all Pull Requests, however, can lead to a resolution. A single Issue can have multiple Pull Requests opened.

Figure 2.3 contains a snippet of a Pull Request submitted in Mockito. It contains some similar contents as Issues, for instance, a title, body, ID, author information and labels. However, it also provides extra information in the form of number of Commits, checks and files changed to convey the breadth of changes contained in the Pull Request. It also shows the source branch where the code is written and the base branch where it will be merged. The status "Merged" shows that this Pull Request has been accepted and the branches merged.

To link a Pull Request to the Issue it is resolving, the Issue's ID is mentioned in the text. For instance, the Pull Request in Figure 2.3 is aimed to resolve Issue with ID 1988.

For repository mining, Pull Request provides the most resources and possibilities with their interconnections. The series of Comments and Reviews provide insight into the collaboration among contributors, and the Commits contain contribution in the form of source code. Being interconnected in Pull Requests, these three artifacts show how contributor collaboration and discussion can directly in-

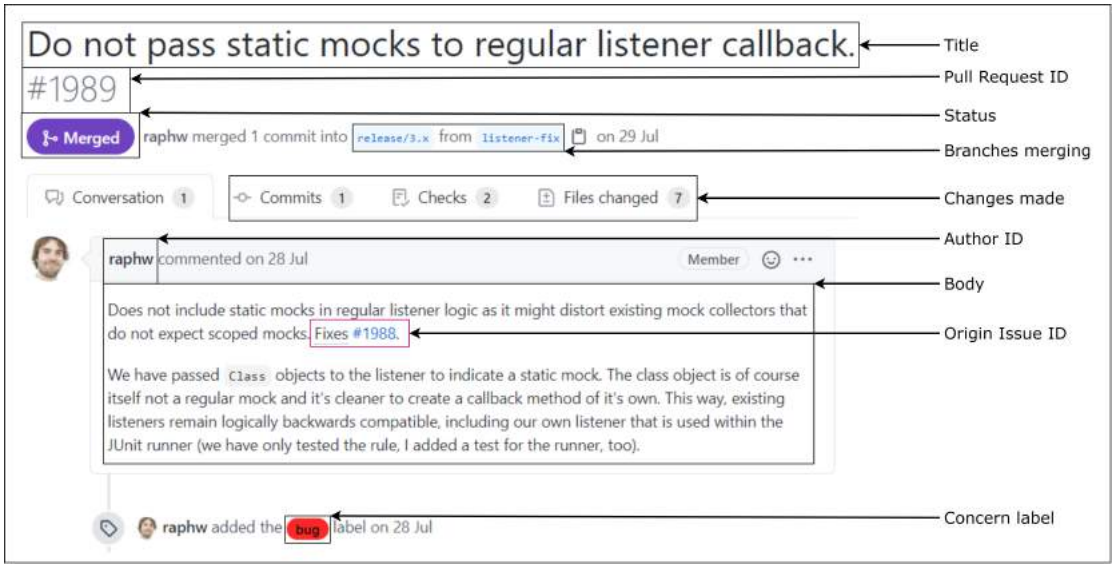

Figure 2.3: A sample Pull Request from Mockito

fluence the contribution made. This study utilizes this inter-linked and resourceful aspect of Pull Requests to conduct its analysis.

2.2.3 Collaboration Processes

Utilizing these artifacts, GitHub accommodates three collaboration processes which this study analyzes. The choice of process is based on the following criteria:

1. Each process enables interaction between a contributor and the project, or among multiple contributors regarding the project.

2. Since this study incorporates sentiment analysis, the processes also generate textual output from the interactions.

3. Lastly, to correlate contributor sentiment to contributor performance, all the processes result in developer code.

The processes are described as follows.

Code Submission

Submitting code is the primary responsibility of developers in a software project. Commits, as described in Section 2.2.2, are the unit of change in a software, containing a cluster of changes to code made by a developer. It is the most direct interaction between a contributor and the software. It generates textual data in the form of Commit messages, as showed in Figure 2.1. The Commit itself is used for quantifying developer performance, as specified in Section 2.3.

Code Review

For code submitted in a Pull Request, it is associated with reviews from reviewers and moderators in the form of comments or review comments. The difference between the two entities comes in their usage and scope.

Comments in Pull Requests are structurally similar to comments in Issues. As demonstrated in Figure 2.4, a snippet of Mockito's Pull Request #1833, Comments follow the body of a Pull Request, containing discussion on the posted changes.

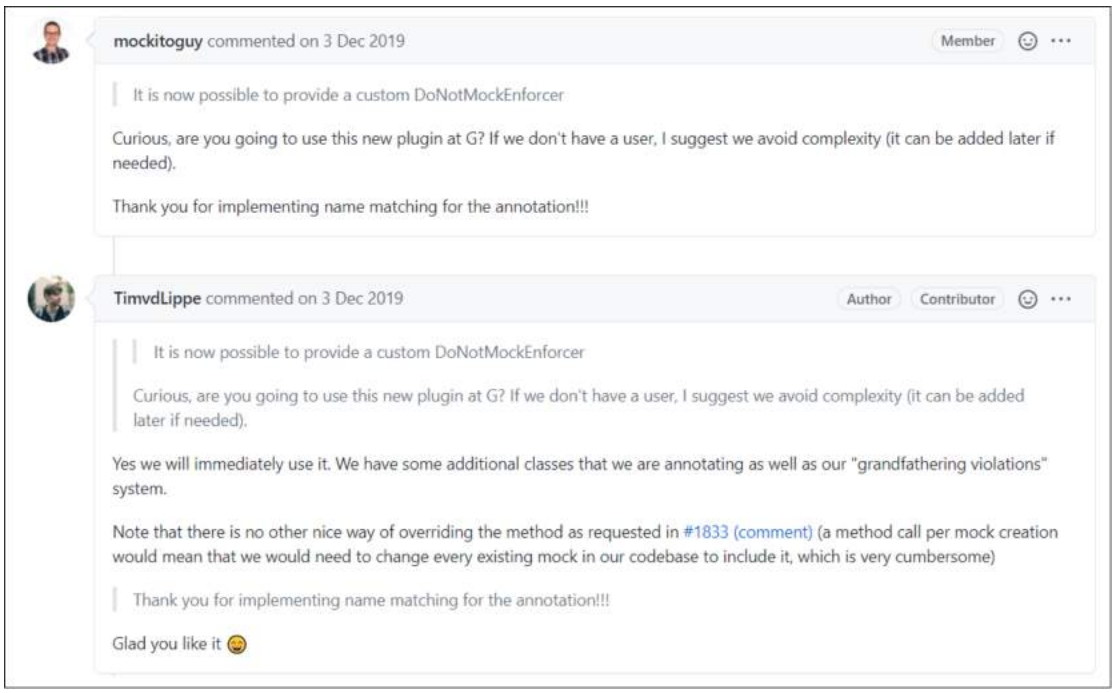

Figure 2.4: Example of a Pull Request comment in Mockito

Review comments in Pull Requests are more code-centric than normal comments. As shown in Figure 2.5, another snippet of the same Pull Request as Figure 2.4, review comments are posted with a segment of the code contained in a Commit. For this example, the review is conducted on lines 84-87 in file *MockitoCore.java*. Reviews create their own threads as shown. The "Outdated" label on the top signifies that this segment of code was later changed.

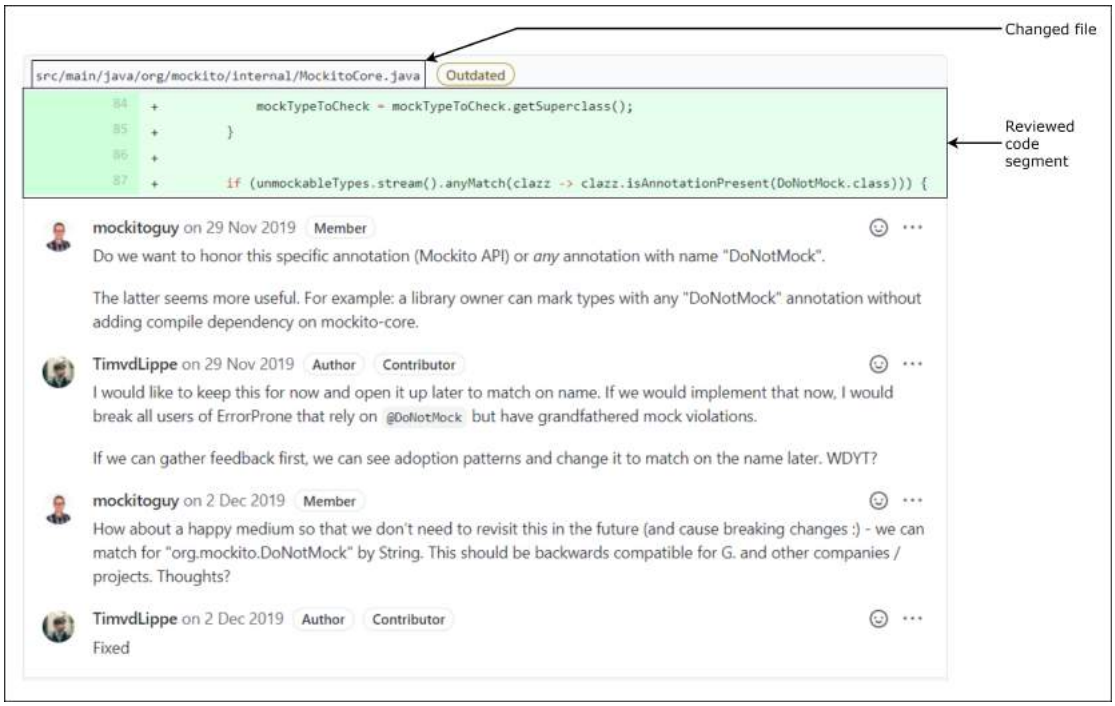

Figure 2.5: Example of a code review in Mockito

Since review comments are directly linked to a segment of the code change, its context encompasses the code more. On the other hand, normal comments contain discussions with a higher level perspective. However, both entities are closely related to the changes made by the assigned developer.

Task Resolution

In this study, tasks are defined as the series of activities starting with a requirement to modify software output and ending in a change of code that satisfies that requirement. The life cycle of tasks contains the following segments:

1. Tasks are initiated once an Issue is posted. The Issue provides information on a problem or shortcoming of the software output as well as a requirement on what the fix would be.

2. Contributors discuss in the comments on the possible way to resolve this Issue. A developer is assigned to change the code in accordance.

3. The assigned developer changes the code to solve the problem and posts Commits with these changes.

4. The developer opens a Pull Request containing the cluster of Commits. This Pull Request is used for reviewing the changes. Comments and review comments are used by reviewers to provide their insights. Based on these, the assigned developer further modifies their code.

5. Once the code is satisfactory, the Commits are merged to the base branch and the changes are incorporated in the software. The Issue is considered resolved and therefore closed, finishing the task.

Tasks contain inter-contributor communication in all its steps and textual output in the form of Issue and Pull Request descriptions, comments, reviews and Commit messages. Since tasks end in changes to the software, the resulting Commits work as a metric to observe how the developer performed based on the previous communications. This life cycle is utilized and analyzed in this study.

2.3 Fix-Inducing Changes (FIC)

A core concept adopted in this study is the concept of Fix-Inducing Changes or FICs [11]. This study adopts FICs to analyze the quality of code — whether a change to the system is problematic or not. Before understanding FICs, an elaboration on software bugs is required. This will enable further discussions into

the changes necessitated by software bugs, including FICs, bug Fixing Changes (FCs) and Fix-Inducing Fixes (FIFs).

2.3.1 Software Bugs

Bugs are a prevalent yet unwanted part of software systems. Code components that deviate the software from its intended outcome are called software bugs [12].

For example, Figure 2.6 demonstrates an instance of a bug. In the method *iterateXTimes()*, the intended outcome is to print the statement, "This is not a bug," a total of x times. Here, x is input as a parameter of the method. However, the *loop* statement is configured incorrectly, which will iterate the function $x + 1$ times instead. Therefore, the method's actual outcome deviates from its expected one. This *loop* statement is considered a faulty code, representing a software bug in the system.

```
string iterateXTimes (int x) {
    string s = "";
    for (int i = 0; i <= x; i++) {
        s.append("This is not a bug.");
        s.append("\n");
    }
    return s;
}
```

Figure 2.6: Example of a buggy code

In this study, bugs are used as a metric of invalid change to the software. Commits that contain bugs are considered as negative contribution. It is regarded as a negative addition to the system because bugs correlate to a waste of project effort, a decline of product quality and consequently, financial losses [13, 14]. An effort to proactively reduce bugs from originating is therefore a necessary part in software development and maintenance processes.

To prevent bugs from being introduced, the circumstances that cause these must be identified. Multiple reasons attribute to such contributions, including ambiguous requirements, unclear documentation, misleading communication and more [78]. This study, therefore, analyzes an aspect of these circumstances: communication and collaboration among the contributors of a software project. More specifically, a correlation between the sentiment in such collaborative elements and the changes related to bugs is explored.

2.3.2 Bug Related Changes

Bugs are associated with different types of changes to the system. Some changes are intended for fixing these while others are categorized as their source. These changes, encased in Commits, work as components for judging developer code.

Bug Fixing Change

Bugs in the system are detected through user reviews of erroneous software behaviour and/or developer-written test cases [79, 80]. Based on these inspections, the buggy code is traced or localized [81]. After localization, a developer is assigned to fix the detected bug. Using one or several Commits, the developer adds patches to the buggy components, repairing the erroneous code. These Commits are regarded as Bug Fixing Changes or Fixing Changes (FCs).

To extend upon the previous example, Figure 2.7 shows a patch to the bug that removes the faulty code. This change is considered as Fixing Change.

In studies assessing and analyzing software bugs, Fixing Changes are essential components for analysis as they provide a gateway to the bug's life cycle. FCs signify the end of a bug's life and from these, the bug's origin can be extracted.

```
string iterateXTimes (int x) {
    string s = "";
    for (int i = 1; i <= x; i++) {
        s.append("This is not a bug.");
        s.append("\n");
    }
    return s;
}
```

Figure 2.7: Example of a buggy fix

Buggy/Fix-Inducing Change

Fix-Inducing Changes (FICs) are changes to the code that cause problems to the software in the form of bugs [11]. Although assigned a different purpose, these changes unintentionally insert faulty code, which end up being software bugs. A bug in an FIC later necessitates another change to the system which repairs the code and removes the bug. Since the buggy change induces a later fix, it is termed as Fix-Inducing Changes.

FICs cannot be directly detected from the archive of Commits of a software project, since no indication of inserting the faulty code can be found in a Commit's description or message. Rather, FCs are utilized to detect those. The message in FCs contains keywords that indicate a bug is being repaired — "bug", "fix", or "patch" [28] — and the code changes documented in FCs indicate the exact lines with faulty code. These two information are used for tracking back and detecting the Commit where this faulty code was originally written, or in other words, the Fix-Inducing Change [11, 29].

In Figure 2.8, Commit B shows the FC where the faulty code was modified and the bug was removed. From its message, it is categorized as a bug fixing Commit. From its line change, the faulty code in the loop statement is detected. Iterating through the archive of Commits, Commit A is traced where this line was

last updated. In doing so, the origin of the faulty code is detected and Commit A is labeled as the Fix-Inducing Change.

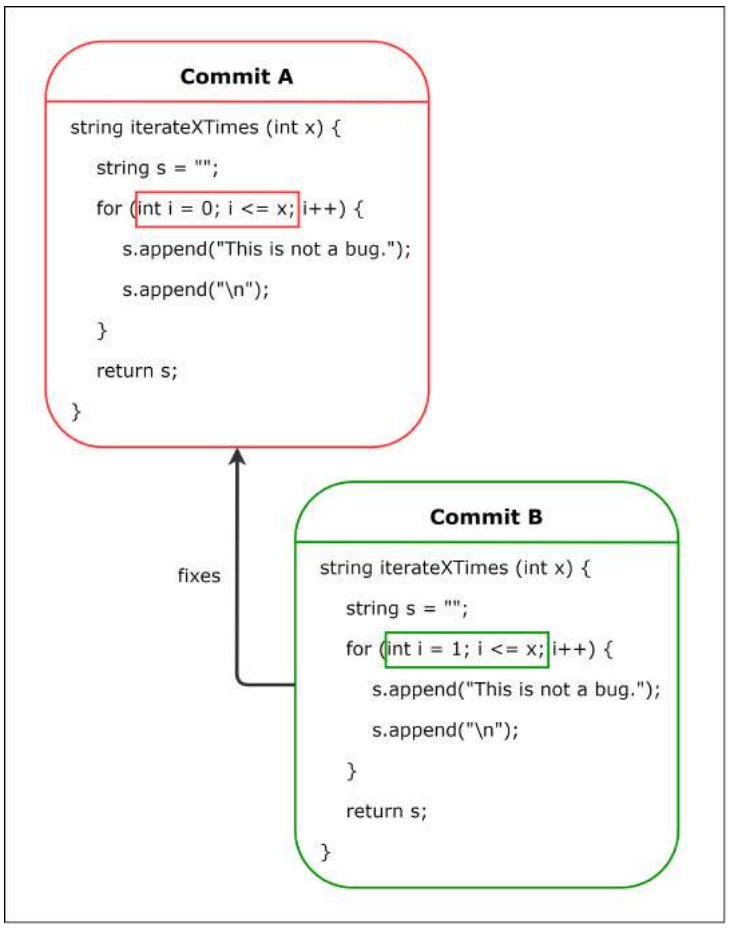

Figure 2.8: Connection between FC and FIC

As software bugs reside strictly in the software code, analysis of software development processes requires a project-level artifact to signify a bug. This requirement is fulfilled by FICs, as these are the Commits that introduce bugs to the system. Since a Commit contains information on the developer who wrote it, the date and time of submission, an associated message [24], and a connection to other artifacts like Pull Request and Issues [76], FICs are essential for analyzing and assessing bugs in the software development paradigm.

Fix-Inducing Fix

The last kind of bug related change that is needed to be discussed are Fix-Inducing Fixes (FIFs), which are a mix of the previous two. FIFs are changes to code that were intended to fix a bug but ended up introducing a different one [29]. In other words, Fixing Changes that become Fix-Inducing Changes are regarded as Fix-Inducing Fixes.

To understand FIFs, the previous example is expanded upon. In Figure 2.9, a new Commit is retroactively inserted between Commits A and B. This new Commit, Commit F, is tasked with remedying the faulty code of Commit A. While it modifies the faulty *loop* statement that caused $x + 1$ iterations instead of x, this new code accidentally causes $x - 1$ iterations because of the faulty logic. Therefore, a later Commit, Commit B, is submitted that remedies this new bug. So Commit F is bug fixing Commit that itself introduced a bug, making it a Fix-Inducing Fix.

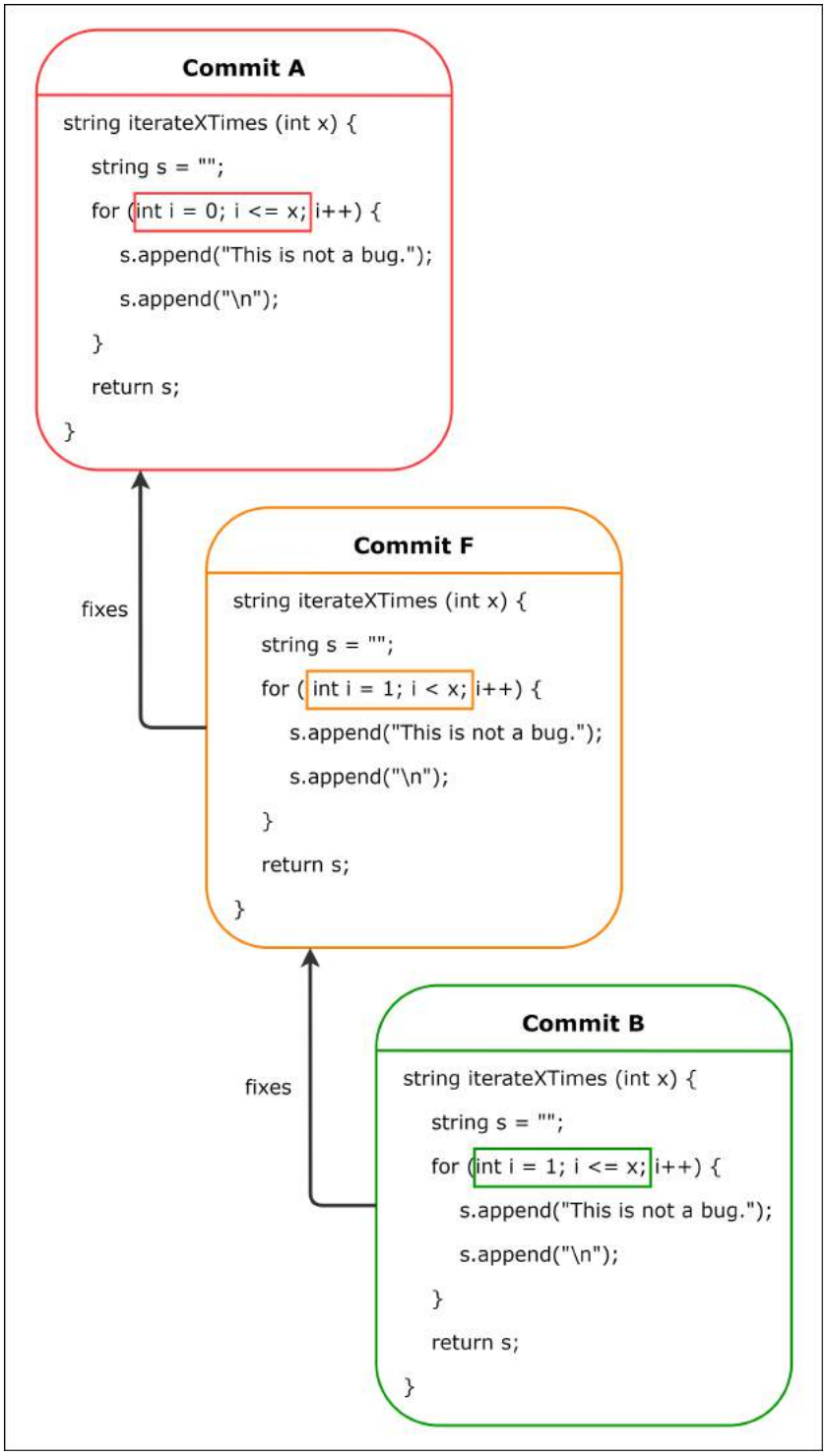

Figure 2.9: An example of FIF

Chapter 3

Literature Review of Sentiment Analysis in Software Engineering

The field of sentiment analysis in software engineering has seen studies that explored the patterns and implications of sentiment in software development processes. Inspired from researches that related sentiment in the workplace with employee behaviour [19, 20], the studies in this field observed relations of sentiment with developers and the development processes. The studies put to use multiple sentiment analysis tools that are both general purpose [82, 83] and catered for the domain [84, 85, 86].

Software repositories — bug trackers, Issue trackers and online code repositories — offer development teams with contents to aid their processes. Communication and collaboration are conducted through these repositories, which store all the information related to the software and its development processes. A storage of direct input from the developers, these repositories are the perfect medium to study the behaviour and sentiment of software teams. Researches utilizing these repositories studied the following relationships regarding sentiment:

1. Project characteristics: language used [5]

2. Task property: day or time of posting [5, 6], size of task [6]

3. Team distribution: geographic location [5], reviewer position [10]

4. Productivity: Issue activity [4], Issue resolution time [7, 8], review resolution time [10], developer inactivity [9]

5. External incidents: deadlines [27], core contributor retirement [9]

6. Task outcome: build success/failure [31]

These studies are described in depth in the following sections grouped based on the artifacts utilized for the sentiment analysis.

3.1 Sentiment Analysis on Compound Artifacts

Early studies in this domain utilized multiple artifacts analyzing contributor sentiment and extracting their influence on the development process. The combination included artifacts extracted from a collaboration tool or repository and a mailing list used for inter-contributor messaging.

The first of these studies by Guzman et al. [27] was one of the earliest to explore sentiment in software development processes. Instead of archetypal open source projects, their analysis was conducted on student projects in collaboration with industrial partners. Their dataset consisted of three projects developed by teams of 5 to 8 students in the span of three months. For sentiment analysis they extracted two categories of textual data:

1. The teams' mailing list, which contained inter-contributor mails regarding the project, and

2. Artifacts from Confluence [87], an online project collaboration tool that can be integrated with software development processes. The artifacts include: Commit messages, bug reports and wikis.

On the extracted 857 emails and 143 web pages from Confluence, they conducted sentiment analysis using SentiStrength [82]. It is a lexicon-based sentiment analysis tool, as described in Subsection 2.1.2. The tool provides sentiment polarity scores to sentences based on the predefined sentiment score of words and phrases. Furthermore, they used Latent Dirichlet Allocation (LDA) [88] as a tool for topic modeling, summarising the textual content to represent their overarching context. An example context can be represented by the word set *bug, problem, crash, application* which convey a bug-related text.

Lastly, they interviewed the project leaders, who were graduate students, on the relevance and practicality of the results. Their findings include:

1. Sentiment values are significantly different in the two sources. The mailing list, which had been a medium for discussing organizational and generic issues, contained emotional messages. On the other hand, artifacts in Confluence, whose primary use was recording meeting protocols and general knowledge of the software being developed, had more neutral text.

2. From the interviews, it was seen that the leaders were able to correlate the flow of polar sentiment with the teams' performance and motivation, and project deadlines.

This early study had been an insightful guideline for future researches, some of which described in this chapter, for conducting sentiment analysis on software engineering projects.

The other study utilizing multiple artifacts was conducted by Garcia et al. [9]. They aimed to understand how emotions correlated to contributor activity. They analyzed the bug repository of the Gentoo project [89] with the following research goals:

1. **Emotional effect of core contributor turnover**: Their selection of the Gentoo project was partially motivated by the sudden departure of the com-

munity's core contributor named Alice. Her retirement adversely affected the structure and processes of the project. The study delved in the emotional patterns of Alice leading up to her turnover and how it affected the sentiment of the other contributors afterwards.

2. **Effects of emotions on contributor activity**: As an extension of their first goal, they aimed to understand whether emotional patterns exist in terms of contributor inactivity. Instead of focusing on a single contributor and her turnover, this next step observes the general contributors and their elongated periods of inactivity or lack of contribution.

For their analysis, they used two disjoint datasets. The first consisted of the bug reports archived in Bugzilla [72]. From the span of 10 years of development, the project provided 661,783 Comments among 36,555 contributors. The second dataset was extracted from the developer mailing list, which was used for exchanging bug fix-related information and requests towards maintainers. The mailing list yielded a total of 81,328 messages. To extract sentiment from the textual data of the datasets produced, they used SentiStrength [82].

Their analysis produced the following findings:

1. In the period before Alice's departure, messages in the mailing list where Alice participated in were significantly more negative than ones where she was not a part of. This could represent hostility between her and other contributors that might also influence her retirement.

2. After Alice's retirement, the project faced intense reorganization efforts. In this period, the mails were more negative in general compared to mails in other periods.

3. Inactivity was measured as zero contribution from a contributors for at least 30 days. To differentiate the emotions between active and inactive states,

each interval was taken and sentiment values 5 days preceding the intervals were calculated. From these values, a prediction model was developed to estimate the likelihood of a contributor going inactive. The two datasets — bug reports and mailing lists — contained two different outcomes:

- In bug reports, polarity was not observed as an important feature when predicting contributor inactivity. Rather, intensity of the sentiment showed to be more valuable, as in a contributor showing strong negative or positive emotions ended up more likely to become inactive.
- In mailing list, the predictive feature was the contributor's emotional deviation. It is seen that contributors who show emotions that divert greatly from their usual expression tend to go inactive from the project.

These results show that large events in a software project emotionally affect the contributors and their mood expressed in collaborative and conversational artifacts can indicate their motivation to further contribute. These two studies paved the way to utilize the textual data found in project artifacts for sentiment analysis. Based on these, the following studies mined certain artifacts and analyzed specific patterns and correlations with the development processes.

3.2 Sentiment Analysis on Commits

Commits are the documented changes to code that contain a direct message from the developer. As described in Subsection 2.2.2, Commits are a core component of the software development process. Commits consist of multiple attributes: Commit message, author information, timestamp, list of changes made to the code. Additionally, Commits are linked to other aspects of the development cycle: originating Issues, reviews and more. These components provide various opportunities for understanding developer and development patterns. Studies of sentiment anal-

ysis on Commits have analyzed the sentiment of Commit messages and observed certain patterns.

The first study in this line was conducted by Guzman et al. [5]. They experimented on 90 projects from GitHub, garnering a total of 60,245 Commits. The goal of their exploratory study was to observe the relationship of the sentiment in Commit messages with multiple development aspects, for instance:

1. **The project language**: the principle language which the project is developed in. Each repository in GitHub contains a list of languages present in the files of the project along with the frequency of those languages. For this study, they used the topmost language of the list.

2. **Time of Commit**: the day of the week and the time of the day a Commit was posted. The timestamp with Commits provides this information.

3. **The team's location**: the geographic distribution with which the development team is formed. Since the dataset consisted of Open Source repositories, the projects consisted of contributors who were mostly geographically dispersed and not centralized. This information can be obtained from a user's profile, which is linked to the Commit via author information.

4. **The project's approval**: star rating the project has in GitHub. For repositories in GitHub, users can provide these with a star as appreciation or praise. The higher the number of stars, the more prestigious the project and renowned the developers' contributions.

To measure the sentiment of the messages, they used the SentiStrength [82] tool. Although previously used by similar studies of the domain, the tool is not context-specific, as in, it is not curated for the software engineering corpus.

The results of their study include the following outcomes:

1. Projects with Java as the principle language contain the most negative Commit messages.

2. Commits posted on Mondays have more negative messages than the rest of the week.

3. The more geographically dispersed a team, the more negative their Commit messages are.

4. A project's approval, or the number of 'stars', has a weak correlation to the Commits' sentiment.

This study was the first one to analyze the existence and pattern of sentiment in developer Commits. The study showed that Commits — the direct contribution of developers — contain different patterns of emotions that can be correlated to the development process and characteristics of the software project. However, whether these patterns have any relationship with how the developers perform or the quality of their code was not explored.

A similar study was later conducted by Sinha et al. [6] on GitHub Commits from a larger dataset. Their dataset consisted of 28,466 projects over the course of 7 years. Their goal was to expand upon the findings of [5] by incorporating a larger dataset of Commits, extending generalizability of the findings. They aimed to extract the following findings:

1. **The general sentiment in Commit messages**: How positive or negative are the developers who post Commits; or whether they show no emotions in such technical matters.

2. **Day of posting**: Similar to the previous study, they correlated sentiment in Commit messages with the day that Commit was posted.

3. **Size of change**: The number of changed files in a Commit. This data is obtained from the list of changes to the code executed in a single Commit.

The list separates changes based on files. This correlation is a step towards correlating sentiment with the metrics of developer contribution.

For this study, they also apply SentiStrength [82] to quantify message sentiment. This enabled direct comparison with the previous study. However, doing so compromised the improved accuracy obtained from software engineering oriented sentiment analysis tools like SentiStrength-SE [84] or Senti4SD [85].

From their analysis, the following findings are observed:

1. Most of the Commits (74.74%) are neutral, showing neither positive nor negative emotions. This is an indication to the lack of emotions conveyed in Commits, which are technical collaborative artifacts. The Commit messages are used for listing in brief the changes made. However, the existence of sentiment shows that developers also use these for including their opinions.

2. Unlike the previous study's results, they observe that it is Tuesday when the Commits are most negative. Despite the difference in the exact days, both results show that Commits posted earlier in the week are more negative. This means developers need time to emotionally cope with the working routine after spending their holidays.

3. They also see that a correlation exists between the size of the project and Commit sentiment. Their last observation shows that Commits with a high number of changed files contain more negativity.

This study delved in the patterns of sentiment in Commit messages with development behavior and project characteristics. Their first observation – neutrality in Commits – yielded a very important information, that Commit messages are mostly formal and opinions are seldom expressed. The correlation between sentiment and number of changed files in a Commit also provided a useful insight. Commits with a high number of files changed are ones where the developer needed

to code on multiple components with multiple concerns at once. However, this pattern does not generate a direct relationship with the quality of the code by the developer. Whether the developer performed poorly for instance, added bugs due to a distinctively different sentiment cannot be deduced from the observations of this study.

The last study of sentiment analysis on Commits was done by Souza et al. [31]. Their study analyzed a more specific aspect of Commits, by correlating the sentiment in their messages with Continuous Integration (CI) builds [90]. CI servers are an added feature to code repositories like GitHub that conducts automated tests and integrates the new changes to the software. These notify whether there are integration issues in the change before merging it and possibly harming the end product.

This study specifically chooses the Travis server as a CI tool, observing the following patterns:

1. **The sentiment of Commits that lead up to a failed build.** Since builds signify correctness of code, it is hypothesized that negative mood may hamper developer performance. This will in turn cause a failed build.

2. **The sentiment of Commits that are posted after a failed build.** The mood after a failed build is hypothesized to be negative as developers are tasked with solving a problem that is pausing other tasks.

3. **The sentiment of Commits mentioning Travis CI.** Observing the sentiment of developers mentioning the server may help discover how Travis is perceived in the community during development tasks.

To conduct this study, the authors used the TravisTorrent dataset [91]. The dataset provided with 1,262 projects in Java and Ruby consisting of a total of 609,467 builds and 1,016,017 Commits. Among the builds, 26.5% were broken.

For analyzing sentiment, this study has used the SentiStrength [82] tool as well. However, due to some context-specific terms like *failure*, *bug*, *violation* and more containing contradictory sentiment, the authors modified the dictionary provided by the tool.

From the analysis, they reported the following results:

1. Broken builds are preceded by Commits with negative sentiment slightly more than positive ones.

2. Broken builds are also followed by negative Commits compared to positive ones, with a small effect size.

3. Travis CI is mentioned in positive messages slightly more than negative.

All their correlations have been associated with a small effect size. This is interpreted as sentiment having a weak influence on the outcome of builds. They have also reported a presence of sarcasm or ironic comments in the Commit messages that cause sentiment to be detected incorrectly.

These three studies have established Commits as a useful tool for extracting sentiment of developers. Each study observed how these sentiment values can be correlated to different patterns and aspects of the development processes. These patterns can be further strengthened by incorporating the performance of developers or the quality of their code.

3.3 Sentiment Analysis on Issues

Issues are reports of bugs found or new features to be added. As detailed in Subsection 2.2.2, Issues can be posted by clients, admins or developers themselves. Following the post, contributors discuss on possible ways to resolve to Issue, that is, fix the bug or incorporate the new feature. The discussion takes place in the

Issue's Comments. Based on the ideas and suggestions of these Comments, the code is generated.

Issues provide a large portion of the textual data generated in development processes in the form of titles, descriptions and Comments. These are also core components in a task resolution activity, as described in Subsection 2.2.3. Hence, Issues have been a popular artifact for analyzing contributor sentiment and correlating it with software development process aspects. Four such studies are elaborated as follows.

Jurado et al. [4] conducted the primary research on GitHub Issues. Their goal was to analyze the existence and evolution of the sentiment latent in Issues and their Comments.

Their dataset consisted of 10,829 Issues, collected from 9 GitHub projects: Homebrew, Bottle, Tornado, IPython, Joomla, Pandas, Ruby on Rails, Raspberry Pi and Scala. They created the dataset using GitHub's API, which provides the artifacts with all the necessary information [92].

For calculating sentiment, they used 4 lexicon dictionaries: Affective Norms for English Words (ANEW) [93], OpinionFinder [94], SentiStrength [82] and WN-Affect (WordNet [83]). Due to the use of different lexicons, each Issue produced sentiment in multiple scales: affective emotions (Anger, Disgust, Joy, Surprise, Fear and Sadness) from WN-Affect and polarity scores (negative, positive, neutral) from the rest.

From the results generated, they first observed the viability of polarity scores in this domain. Pearson's correlation and p-values from a one tailed t-test show that, for polarity scores, there is a significant correlation between positive and negative scores. Due to the ambiguity this causes, they discarded the polarity scores from further analysis, and only analyze the affective emotions. For the affective emotions, they observe the following:

1. **The ratio of emotions in projects**: A majority of Issues do not contain

any emotions (neutral sentiment). Among the affective emotions, *joy* is the most dominant, followed by *surprise*. The dominance of *joy* has been rationalized by the self-inclination of developers to contribute to open source projects instead of being forced to. The negative emotions – *anger*, *fear*, *sadness*, *disgust* – are less in numbers. However, according to Garcia et al. [9], the text with negative sentiment holds more insightful information than positive ones.

2. **The evolution of emotions in projects**: The ratio of emotions in Issues evolves consistently throughout the project's life. However, there are time ranges where a single emotion can evolve quickly, that is, have more instances than in other ranges. These periods of emotional dominance can hold insight into or represent the contributors' response to certain features or tasks. This is similar to the findings of Souza et al. [31], where it is seen that Commits after a failed build show more negative messages than positive ones.

3. **The implication of certain emotions**: Negative emotions like *fear* and *anger* in Issues can indicate potential problems. For example, Figure 3.1 shows an Issue from the Pandas project connoted as *fear*, informing of problematic solution.

```
Title: Add key to sorting functions

Body:
Many python functions  (sorting, max/min) accept a key argument,
perhaps they could in pandas too.

*The terrible motivating example was this awful hack from [this
question](http://stackoverflow.com/a/17157110/1240268)....
for which maybe one could do*

    df.sort_index(key=lambda t: literal_eval(t[1:-1]))

*This would still be an awful awful hack, but a slightly less awful
one*.
```

Figure 3.1: A sample Issue showing *fear* from the Pandas repository

This study provided insight into the existence, evolution and implications of

emotions in GitHub Issues. The next step to their findings, as they have also mentioned, is to relate these emotions with metrics that indicate the outcome of the project, for example, team productivity, individual and group task quality etc. Their study also did not incorporate sentiment lexicons that were catered for the software engineering domain, opting only general purpose dictionaries. These resulted in vague results for the polarity scores, due to which they discarded sentiment polarity altogether. Implementation of domain specific sentiment analysis tools can prove to perform better for Issues and related artifacts.

The next study analyzing the sentiment of Issues is conducted by Ortu et al. [7]. Instead of GitHub, their Issues were collected from the Jira Issue Tracking System [71]. They aimed to understand the impact of contributor sentiment on Issue resolution time. Predicting Issue resolution time is an important aspect of software development processes. An accurate prediction is beneficial to a more efficient task assignment, prioritization and release planning.

Their dataset consisted of the 14 projects in Jira with the highest amount of Comments. The Issues ranged from 2002 to December 2013. Combined, the Issues contained more than 560K Comments.

Their study aimed to observe three aspects of sentiment in Issues:

1. The correlation between emotions, sentiment and politeness
2. The impact of developer affectiveness on the Issue fixing time
3. The individual impact of affective metrics to best explain Issue fixing time?

For the last two goals, they developed a logistic regression model. The model classifies whether the Issue fixing time is short or long. As independent variables they used two types of data:

1. **Issue characteristics**: Giger et al. [95] proposed a list of 8 variables that make up an Issue. These are: reporter's and assignee's previous number of

comments, Issue priority, type, number of watchers, number of developers, number of status changes and number of Comments.

2. **Affective metrics**: From Issues, artifacts can be used to extract multiple forms of sentiment. These are: average sentiment, average politeness, proportion of *love*, *joy*, *sadness* and *anger* in Comments, the title's sentiment, politeness, and the first and last Comments' sentiment and politeness.

After selection, they filtered the variables using a hierarchical modelling approach. In the approach, the model is fed each variable one by one, its performance evaluated against the previous one using ANOVA test, and the variable is filtered out if its addition does not cause a significant improvement to the model.

To understand the impact of individual metrics or variables, first, a baseline output is generated by providing the median of all filtered variables to the regression model. Then, each metric k is considered to calculate the relative increase in performance, as shown in equation 3.1.

$$\frac{(metric\,k\,output\ -\ baseline\,output)}{(baseline\,output)} \tag{3.1}$$

Their results yielded the following findings:

1. There is a weak correlation between politeness and sentiment.

2. Affective emotions are significant for indicating Issue fixing time. Their method produced a precision and recall of 0.67.

3. Positive emotions such as *joy* and *love* are related to short resolution time while negative emotion like *sadness* is related to longer ones.

These findings showed that the sentiment in software development artifacts impact the productivity of a team. This establishes the applicability of the existence and patterns of sentiment that the previous studies detected. However,

this study uses time as a metric for quantifying the productivity, which does not necessarily indicate developer performance.

The last study on Issues is conducted by Mäntlyä et al. [8]. They have mined Issues from Jira as well, analyzing the sentiment in the Comments. Instead of affective emotional states, their sentiment analysis adopted the Valence-Arousal-Dominance (VAD) framework. As described in Subsection 2.1.1, VAD is a dimensional sentiment analysis approach. As opposed to the discrete values in affective emotions like *joy*, *sadness* and more, the dimensions in VAD are not mutually exclusive and brings out more complex human emotions and moods. The authors aimed to utilize these complex dimensional emotional states to understand developer activities and development processes. For calculation, they used the dataset presented in Warriner's lexicon [96].

They have correlated the VAD values with multiple aspects in Jira Issues, which include:

1. **Relation between VAD and Issue characteristics**: How each of the VAD values relate with distinct properties of Issues. Their hypotheses include: high priority Issues contain high Arousal, Issues about defects contain low Valence, Issues about feature additions or improvements contain high Valence, and short Issues contain high Dominance.

2. **Evolution of VAD during Issue resolution**: How the VAD values change during the lifetime of an Issue. Since Issues can take a long time and consist of conversations between reporter and contributors, emotions may differ based on time. Hypotheses include: low Valence at the start of bug reports and high Valence near the resolution of said bug.

3. **Correlation between VAD and defect fixing time**: How well does VAD predict Issue resolution time. This includes comparative analysis with other metrices such as affective emotions, sentiment polarity and politeness.

4. **Impact of Issue characteristics on VAD**: How the activities and properties in Issues can influence the VAD conveyed in the conversation. As a mirror to their first goal, they aimed to identify properties that cause distinct VAD values, for instance, high and low Valences.

To analyze Issues, they collected 700,000 Issue reports from 1000 Jira projects. From each Issue, their title, description and discussion were extracted, amounting to a total of approximately 2 million Comments.

The following findings were observed from their analysis:

1. Arousal scores were compared among the 5 priority levels: *Trivial*, *Minor*, *Major*, *Critical* and *Blocker*. They have found that Arousal scores increase as priority level increases, although with low effect sizes. However, in the case of *Trivial* and *Blocker* Issues, the latter has a higher Arousal with a small effect size.

2. Valence scores were compared among the most popular Issue types: *Bug*, *All Tasks* and *Future Dev*. It was observed that Valence is lowest for Issues of *Bug* type. This finding is congruent with their hypothesis, where they assumed that developers would be less keen on fixing program errors than working on software features.

3. Dominance scores were compared between Issues with high and low resolution time. It was seen that, contrary to their hypothesis, Dominance was significantly higher in longer Issues than shorter.

4. The evolution or change of VAD scores were observed in Issues with at least four Comments. It was seen that, throughout the lifetime of an Issue,

    - Valence increases,
    - Arousal slightly decreases, and

- Dominance increases.

5. In terms of contributor roles and their respective VAD scores, there are separate patterns for the separate roles: assignee, reporter and other commenters. For assignees, Arousal drops near Issue resolution — an indication that task completion decreases emotional intensity. Assignees experience the least amount of Valence among the three roles, which represents assignees doing the actual work while the rest receive the Issue resolution as a "gift". For both reporters and other commenters, Valence and Dominance increase with time. However, while the reporters' Arousal remains stable, other commenters experience a decrease.

6. A logistic regression model was developed to observe the impact of VAD scores on Issue resolution time. It was found that VAD metrics are statistically significant for predicting the time required to finish an Issue. Among the different scores, Valence in title and Comments shows the largest impact.

7. For the last observation, the authors calculated the impact of different Issue properties for predicting the VAD scores in the Comments. One example from the correlations regards burnout — low Valence and Dominance, and high Arousal. It was observed that high priority bug Issues with a high resolution time, multiple watchers and Comments, and an experienced reporter are likely to cause burnout.

The results from this study explored in depth the many ways developer emotions and Issue processes influence each other. The three studies together establish developer sentiment as an integral part of development processes, especially in the context of Issues. These also show the predictive values of sentiment in terms of Issue resolution time, or in other words, developer productivity.

3.4 Sentiment Analysis on Code Reviews

The final study, conducted by El Asri et al. [10], analyzes code reviews, the sentiment intrinsic in the review comments and its effects on the review process. The goals of their study include the following:

1. **Performance of sentiment detectors**: As sentiment analysis is context-dependent [62], tools that detect sentiment perform differently on different contexts or text corpora. Hence, the authors first evaluated the applicability sentiment detection tools commonly used in the software engineering tool on code review texts. The tools include: Senti4SD [85], SentiCR [86] and SentiStrength-SE [84].

2. **Prevalence of sentiment in code reviews**: In the next step, they analyzed the sentiment in code reviews to extract patterns. Their goal was to understand which type of sentiment occur the most in reviews and how these differ based on the reviewer's role.

3. **Effect of sentiment on review time**: Lastly, they checked the impact of the extracted sentiment on the review process. To qualitatively evaluate the reviews, they used the time to resolve the review as a metric.

The dataset of code reviews was collected from the Gerrit reviewing system [97]. It contains four large projects: Openstack, Eclipse, Android and LibreOffice. From the projects, a total of 317,373 reviews and over 4 million Comments were extracted after preprocessing.

Their analysis yielded the following findings:

1. Comparatively analyzing the three sentiment detection tools showed that these perform closely when detecting sentiment in code reviews. However, Senti4SD [85] performs the best with a 79% precision and f1-score. Therefore, the authors use this tool for their following analyses.

2. They observed a prevalence of neutral comments in reviews. Regardless, positive and negative reviews exist, indicating emotional expression.

3. As a reviewer matures in the project's environment, their reviews become more neutral.

4. The difference in sentiment pattern was also apparent in the contributor's position in the peer reviewing network. Core contributors show less emotional messages than peripheral contributors.

5. They found that sentiment in code reviews is correlated with review time. Positivity in sentiment was seen to be inversely proportional to resolution time. It was observed that reviews containing positive comments lead to a reduction in review time by 0.4 day.

6. Lastly, they reported a differing correlation between sentiment and time in reviews based on reviewer position. Review times were consistently longer in the presence of negative comments from peripheral contributors compared to core ones.

These findings provide insight into the patterns and properties of sentiment in code reviews. Additionally, they show that the expressed emotions affect the tasks by developers.

3.5 Summary

This chapter discusses in depth the studies that previously mined software engineering repositories for sentiment analysis. Various artifacts have been analyzed in these studies, including mailing lists, Commits, Issues and code reviews. From these analyses, different patterns and correlations regarding developer sentiment have been explored.

Guzman et al. [27] extracted the flow of developer sentiment on three student projects using collaborative artifacts like Commit messages, bug reports, emails and more. The results show that the sentiment in the artifacts are positively correlated with real life developer behavior. Real life events have also been observed to affect open source software projects. Garcia et al. [9] analyzed the Gentoo project and found developer turnover and inactivity to be affected by extreme and outlier emotional expression.

Mining the Commit messages in numerous open source projects yielded relations between sentiment and various properties of developers and the project [5, 6]. These properties include time and day of Commit, number of changed files, project language, geographical distribution of the team and project rating. Additionally, Souza et al. [31] observed that failed builds in continuous integration systems are correlated to developer sentiment. These studies indicate the effects and applicability of the sentiment found in developer contribution itself in the form of Commit messages.

Next, Issues, which are important artifacts for task management in software development, have been used for conducting sentiment analysis. Issues are observed to consist of developer discussions rich with emotions [4]. Mining efforts on Jira issues have shown that positive emotions are linked to shorter issue resolution time [7, 8]. These indicate a correlation between sentiment found in contributor discussion and developer productivity.

Lastly, code reviews have been analyzed for contributor sentiment by El Asri et al. [10]. It was observed that the sentiment in reviews are correlated with the time taken to develop code based on the review. This indicates the impact of sentiment on the developer's development activity.

So far, the studies in the domain of sentiment analysis in software engineering have established the existence of contributor sentiment in various project artifacts, including Commits, Issues and code reviews. Patterns have been observed along

with different correlations with project properties and developer processes. However, none of the studies have yet correlated sentiment with the quality of code developed, which is a metric of developer performance. In the following chapters, GitHub artifacts are mined, sentiment from their text is extracted and is correlated with Fix-Inducing Changes [11] to explore the impact of sentiment on developer performance.

Chapter 4

Sentiment in Developer Contribution and FIC

The outcome of software products primarily depends on the developers who build these with their code and design decisions. The activity of developers is affected by their emotion or sentiment in a software development environment [32]. The influence of developer emotions has been observed in previous studies that correlated these to several aspects and patterns related to the development process. The patterns include task resolution time [7], developer turnover [9], etc. that are negatively affected by developer sentiment. The sentiment of developers can be extracted by conducting sentiment analysis on software collaborative artifacts. Among the many collaborative artifacts, this chapter focuses on Commits [24], which are virtual documentation of a developer's contribution. With the sentiment quantified from Commits, a relation between developer sentiment and software bugs is observed. To do so, Fix-Inducing Changes [11] — changes that introduce bugs to the system — are detected, along with changes that precede or fix those bugs. Sentiment of these changes are determined from their Commit messages using Senti4SD [85]. It is statistically observed that Commits that introduce, precede or fix bugs are significantly more negative than regular Commits,

with a higher proportion of emotional (non-neutral) messages. It is also found that there is a distinction between buggy and correct fixes based on the message's neutrality. This result indicates that a relationship exists between the sentiment of developers and their activity that causes bugs.

4.1 Introduction

Sentiment analysis is the process of extracting human emotion from written natural text [1]. It is an important approach for understanding the behavioural pattern of single or multiple individuals over a sizeable set of text. Its necessity emerges when the dataset is too large to evaluate the input manually. Therefore, it can be applied in the software development environment, where, in the span of the project's lifetime, the developers and contributors add their valuable textual input that shape the software. Software developers post their textual input in online collaborative artifacts:

- Commits — developer code
- Issues — contributor discussion or client input
- Pull Requests — code reviews

Among these, Commits store historical data on the code changed by developers. Working as a unit of change to the software, a Commit encompasses a specific independent task assigned to a developer. The components of a Commit is displayed in Figure 4.1. Commits are assigned a unique SHA or hash that works as an identification code. Each Commit contains information of who and when changed the code, along with the change history — lines that were added, deleted or both. Lastly, the developer or author of the commit adds a message to the Commit. This message is written to summarize or describe in natural text the changes they

have made in this particular Commit. This message can be used to extract the developer's sentiment regarding the Commit's task. Commits, therefore, work as an important tool in establishing a direct relation between developer's performance and their emotion.

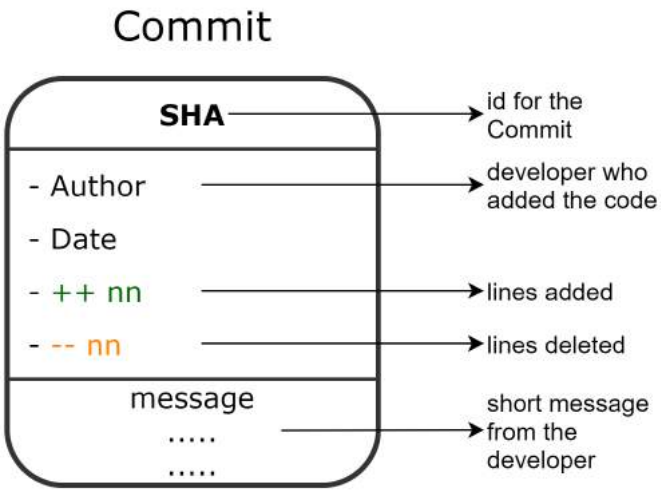

Figure 4.1: Components of a Commit

Alongside sentiment, Commits can be used to derive developer performance by understanding the nature of the changes made in these. Commits can be categorized based on the assigned task. Tasks can range from bug fixes and refactoring to adding new features or improving existing ones. Commits can also be categorized based on the type of code written in that Commit. For instance, buggy code can be extracted from Commits by finding Fix-Inducing Changes (FIC). FICs are Commits that induce fixes or, in other words, cause errors in the system [11]. FICs can be detected using the change history stored with Commits.

In this study, the concept of FICs is adopted to categorize Commits based on bugs. Firstly, the FICs themselves work as the central unit. These are regarded as Commits that introduce bugs. Next, Commits that fix bugs are taken as Fixing Changes (FCs). These are Commits that are conducted as part of an assignment that aims to remedy an error in the system. FCs remove or modify the changes introduced by FICs. Hence, FCs eradicate bugs. Another aspect of

this categorization is the parent of FIC (pFIC), which is regarded as the Commit on which the FIC's changes were implemented. A graphical representation of this categorization is provided in 4.2.

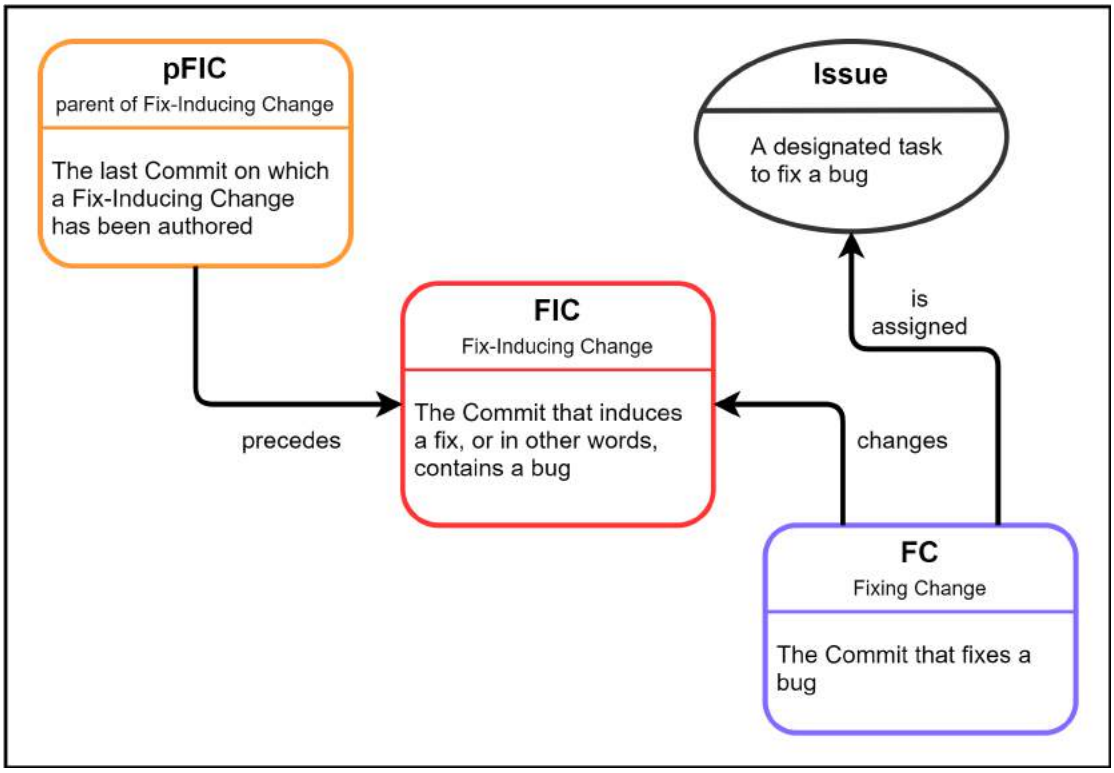

Figure 4.2: Commit categorization based on FIC

Messages in these Commits can contain different emotional expression than other Commits, to indicate complex assignments or imperfect work. Commits related to bug introduction or fixing provide insight based on the messages' emotion. Therefore, this study observes the emotional patterns of four types of Commits to understand the relation between developer emotion and software bugs. With this aim, the Research Question 1 **(RQ1)** is answered through the following four Sub-Research Questions (SRQs) are answered:

SRQ1: How does developer sentiment relate to Fix-inducing Changes? FICs are regarded as Commits where bugs originate. In research, these are used as units of instances where bug is written. Hence, these are utilized to understand the nature of the instance rather than the code.

FICs have been analyzed to correlate the introduction of bugs with various other project metrics. Sliwerski et al. [11] showed that changes in FICs are larger, covering more files than normal changes, and changes made in Fridays are more likely to introduce bugs. Bavota et al. [98] examined whether code smell refactoring cause bugs and found that refactorings related to hierarchies are correlated to FICs. Sadiq et al. [30] experimented with FICs to understand the effect of change couplings.

These studies have proved how effective FICs are in understanding different aspects of the development process. Furthermore, new opportunities for experimentation are also opened. With that opportunity, this study aims to understand the correlation between FIC and developer sentiment; whether the emotion conveyed in the message of a Commit can indicate the negative nature of its changes.

SRQ2: How does developer sentiment relate to the parent of Fix-inducing Changes? The parent of a Commit is the Commit that immediately precedes it. The concept of precedence is based on Git's working tree instead of time or author. Hence, the parent Commit can either be written by the same developer or a different one, and there can be large time lapses between the two. Only the Commit which exists atop the current one in the working tree is regarded as the parent.

The parent contains the latest code on which the current Commit was written. Hence, the parent is responsible for the state of the code before the changes of the current Commit are initiated. To understand this state, the current developer refers to the parent's message which describes the context of that code. The parent of an FIC (pFIC) is important for sentiment analysis as the coder of FIC refers to its message and can be influenced by it.

SRQ3: How does developer sentiment relate to Fixing Changes? Fixing Changes (FCs) are the Commits that fix bugs introduced by FICs. These Commits are designated tasks for removing bugs. The designation occurs through

issue or bug reports posted by the end-user, client, project manager or another developer. The report contains high level specifications regarding the bug or fault followed by discussion among the contributors to narrow down to possible solutions. Afterwards or during the discussion, a developer is assigned the bug report. Then it is their responsibility to fix the bug, based on previous Commits and the report discussion.

Bugs are one of the topmost issues reported in software projects [99, 77] and fixing these is an integral element of both the development and maintenance phases of the project. Extensive research efforts have been put into the nature of bug fixes, how these affect the software process and vice versa, how much effort-time and calendar time is required to resolve these and more. These have been used for predicting issue resolution times. Multiple factors, for instance, bug location in code, reporter of bug and attached description have been analyzed for prediction [100]. Alongside these factors, sentiment can be an important aspect. However, to the best of our knowledge, no such studies have yet been performed to correlate bug fixing activities and contributor sentiment.

This SRQ aims to understand the developer's sentiment when fixing a bug. Sentiment in their messages can provide insight into the developer response to such tasks. A correlation can prove helpful in future analysis of bug fixing activities, by incorporating related emotions in prediction models.

SRQ4: How does developer sentiment relate to Fix-inducing Fixes? When FCs themselves create new bugs, those Commits are regarded as Fix-inducing Fixes (FIFs) [29]. FIFs can simply be regarded as incorrect patches to previous bugs. In their study on the characteristics of incorrect patches, Yin et al. [29] observed that upto around 25% of their sampled fixes were faulty, creating future issues for the software product. They also analyzed the human factors of faulty patches. Their observation showed that developers and reviewers assigned a patch needs to be knowledgible about the section of code base where the bug

exists in. About 27% of incorrect fixes occur due to their lack of knowledge, where the developer have not interacted with the code before.

The aim of this SRQ is to expand on the human aspects of incorrect fixes by incorporating developer sentiment. Sentiment extracted from the Commit messages will be differentiated between correct and incorrect fixes to establish a correlation. A correlation between sentiment and incorrect fixes will bolster the need to monitor the developers' state of mind alongside their proficiency on the bug.

In order to correlate sentiment with bug-related Commits, this study first collects Commits of GitHub repositories and categorizes these based on the SRQs into four different types: FIC, pFIC, FC and FIF. Sentiment values for the Commit messages are extracted using Senti4SD [85], a sentiment polarity classifier specialized for the software engineering domain. Lastly, statistical analysis is conducted on the resulting sets of sentiment values with that of regular Commits.

The findings of the statistical test show a significant relation between bug-related changes and sentiment. All four Commit types are, on average, more negative than regular Commits. Additionally, neutrality is observed to be more predominant (6%) in regular Commits. In terms of polarity, pFIC, FC and FIFs have 12% more negative Commits than positive. Lastly, FCs that have more negative messages tend to become FIFs.

4.2 Study Design

This study aims to incorporate sentiment analysis with the introduction and patching of bugs. To do so, Commits are extracted from open source repositories and analyzed to detect the four intended types. Sentiment analysis on the Commit messages are then conducted to extract developer emotion. The methodology described is conducted on thirteen different repositories from GitHub. The processes are described in the following subsections.

4.2.1 Commit Categorization

The four Sub-Research Questions (SRQs) involve four types of Commits: Fix-Inducing Changes (FIC), Parents of FICs (pFIC), Fixing Changes (FC) and Fix-Inducing Fixes (FIF). This categorization is based on the changes' involvement with bugs.

- FICs contain code that is or causes bugs.
- This buggy code is posted right after pFIC, the closest parent to FIC and the code on which the bug was written.
- FICs also prompt FCs, code that removes the buggy parts introduced by FICs, as a designated task to patch a reported bug.
- Lastly, changes, that are assigned with removing a bug but in doing so creates their own, lie in the category of FIFs.

Commits are stored as part of a project's version control history. Hence, these can be extracted from software projects that adopt a version control system, for instance, Git. From the extracted Commits, the mentioned categorization concept can be conducted computationally. To automatically detect and categorize all four Commits, the following steps are applied.

1. Remote repositories of open source projects from GitHub are cloned locally. Amongst the multiple branches, the default or "master" branch is chosen for analysis. Only Commits from the default branch are extracted since the contributions from different developers are reviewed and merged here. On the other hand, Commits in the other branches do not necessarily represent the final product and may contain intermediary code that do not make it into the production code based unchanged. Hence, even if those code are buggy, those cannot be tracked.

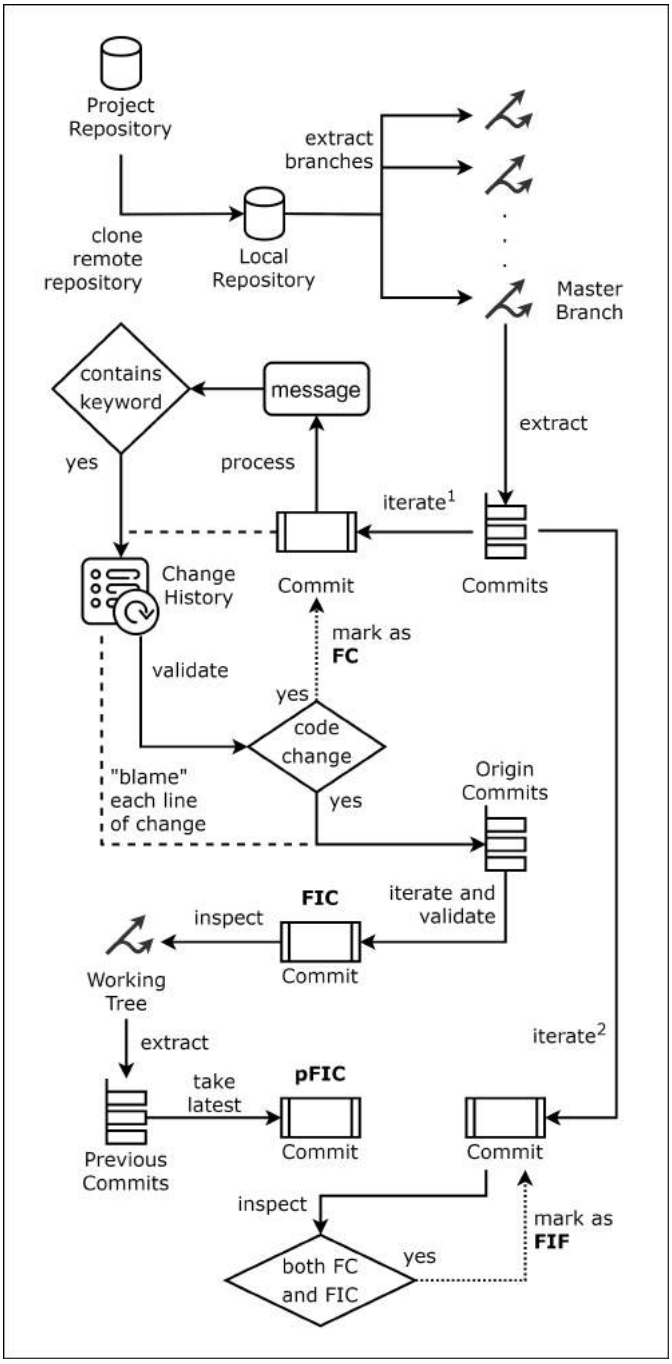

Figure 4.3: Commit categorization methodology

2. Iterating through the Commits, each Commit's message is extracted. The messages are analyzed to understand the purpose or contribution of the Commit. The message text is lemmatized to create a bag of words. This helps in distinguishing certain words or terms. From the lemmatized text, terms like "bug", "fix" and "patch" are searched. The presence of these words indicates that the Commit is related to a bug fixing activity [28].

3. Next, the modifications in these Commits are analyzed to observe the exact changes. Each line in this Commit that is changed from its parent Commit is listed. It is checked whether only non-code changes like documentations and comments occur. Non-code changes do not affect the end product's processes and performance. Since these cannot introduce bugs, no bugs can be removed by modifying these. Furthermore, the type of change is analyzed. There are three types of changes in Commits:

   - Insert: new code is added as a new line
   - Delete: a previous line is deleted wholly
   - Replace: new code is added by modifying a previous line

   Only Delete and Replace types are considered because these indicate the removal of buggy code. This proves that a previous Commit exists where that code was introduced, which can be tracked and extracted. After these two filtering mechanisms — existence of source code modification and the modification not being solely Insert type — the selected Commits are categorized as **FCs**.

4. In the next step, the modified lines in the Fixing Changes are compiled. From these non-code and Insert changes are excluded. With the remainder, their origin Commits are tracked. The tracking utilizes Git's "blame" functionality, which provides the history of a specific line of code throughout

the documented working tree of the project. For each line in FCs, which signify code that fixes bug, a list of Commits where that line was modified is extracted. These Commits signify the introduction of the buggy codes that FCs patch. The Commits found from this step are labeled as **FICs**.

5. After detecting FICs, the working tree is once again used. This time, instead of tracking based on changed lines, the tree referenced based on subsequent changes. For each FIC, its immediate parent Commit in the working tree is extracted and labeled as **pFIC**.

6. After the first iteration of finding FCs, FICs, and pFICs, a second iteration through the processed Commits is conducted. From this iteration, Commits that are labeled as both FC and FIC are categorized as **FIFs**.

A summary of this process is graphically demonstrated in Figure 4.3. From the extracted Commits from the Master branch, the two iterations are labeled as iteration 1 and iteration 2. The first one leads to the detection of FC, FIC and pFIC. The second one, starting after iteration[1] finishes, detects FIFs.

Application of this process is displayed in Fig. 4.4 where a series of Commits on a single file is analyzed. Commits 1 to 9 are extracted using step 1. Beside each Commit, information on the code changes is provided. "++nn" means that new code is added on line nn. Similarly, "–nn" demonstrates the removal or modification of code in line nn.

From steps 2 and 3, the changes of each Commit are extracted and analyzed, leading to the detection of FCs. These are represented as the dashed Commits — #4, #8 and #9.

Next, for each of these Commits, step 4 is conducted. First, it is seen that #4 modifies lines (134, 139). These are introduced in Commits #2 and #3. Hence, these two Commits are labeled as FICs. Their parents, #1 and #2 respectively, are pFICs.

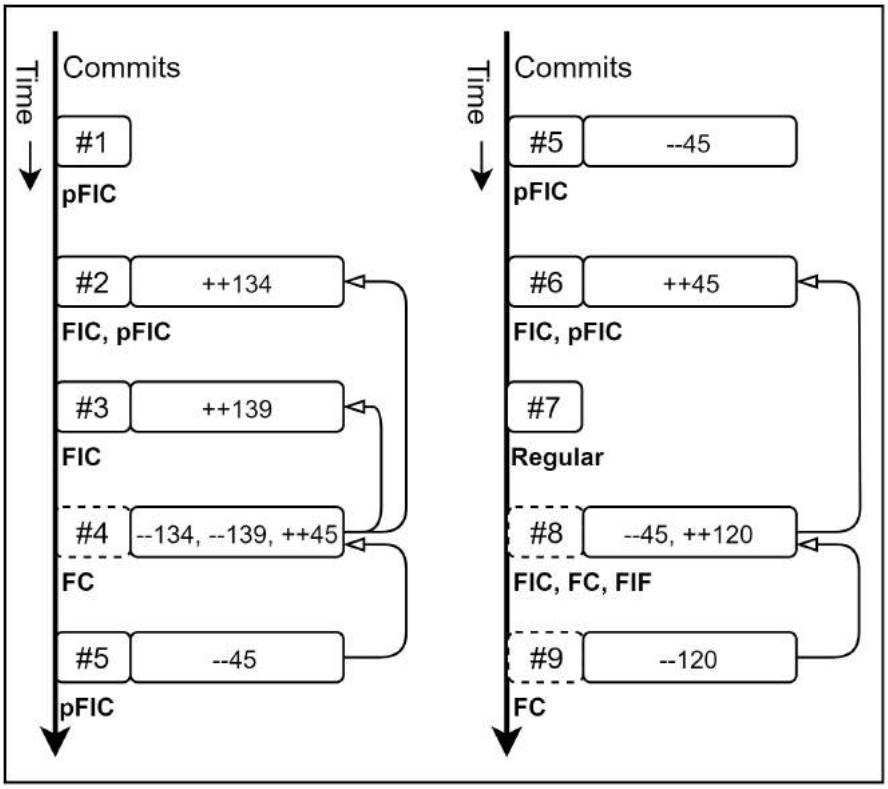

Figure 4.4: Detecting and categorizing Commits

Now, although #5 removes a code (45) of #4, since #5 is not an FC, #4 is not labeled as an FIC. The change can be attributed as a modification in the documentation or changes to the code that do not correspond to the change of functionality, for instance, refactoring activities.

Next, it is seen that the FC #8 modifies line 45, which is introduced in both #4 and #6. However, only the latest update is considered, hence #6 is the FIC.

Lastly, for FC #9, the FIC is similarly inferred to be Commit #8, which adds line 120, removed by #9. This concludes the first iteration. In the second one, Commits that are both FC and FIC are searched. In this scenario, only #8 is such a Commit, hence it is labeled as an FIF.

4.2.2 Quantifying Sentiment

After categorizing the Commits, their sentiment values are quantified. The sentiment extracted from the messages provides two insights:

- subjectivity (neutral vs emotional): whether a text conveys any emotion or simply states a fact
- polarity (negative vs positive): for emotional text, whether the sentiment conveyed is positive — a compliment, praise, joyful remark etc — or negative — insult, criticism, angry remark etc.

This study deals with both of these aspects along with the sentiment's numeric value. When quantifying sentiment, two factors affect its correctness: the context of the words and their intrinsic emotional value.

Context refers to the nature of how word is used in a sentence. To understand the context, the meaning of the text and its parts needs to be approximated. Hence the text is not treated as a bag of words, which only compiles the different words used in their base form. Rather the inter-relation of the words, clauses and sentences are interpreted.

Secondly, while for most words, their inherent sentiment value is universal, some words can have different meaning based on the domain it is used in. In the software engineering domain, words and terms like "bug", "errors" etc do not have the traditional negative connotations. Although these are terms to denote negative aspects of the software, their use is not emotional, rather factual. For instance, in an issue report, a developer can mention that a new bug has been detected in a certain component. Their statement conveys no emotion, rather the objective truth that a component contains code that is causing unintended behaviour in the software. These exceptions must be handled to extract the correct sentiment values of the Commit messages.

In this study, sentiment values for the Commit messages are extracted using the Senti4SD tool [85]. It is an emotion polarity classifier specializing in the software engineering domain. It is used as it eliminates the two correctness factors mentioned previously.

- The tool uses lexicon-based and keyword-based features along with semantics to better contextualize the words.
- It has been trained on a gold standard of 4423 StackOverflow posts, which incorporates software engineering domain-specific terminologies.

The tool has been observed to perform best among similar sentiment analysis tools when conducted on collaborative artifacts in GitHub [101, 10].

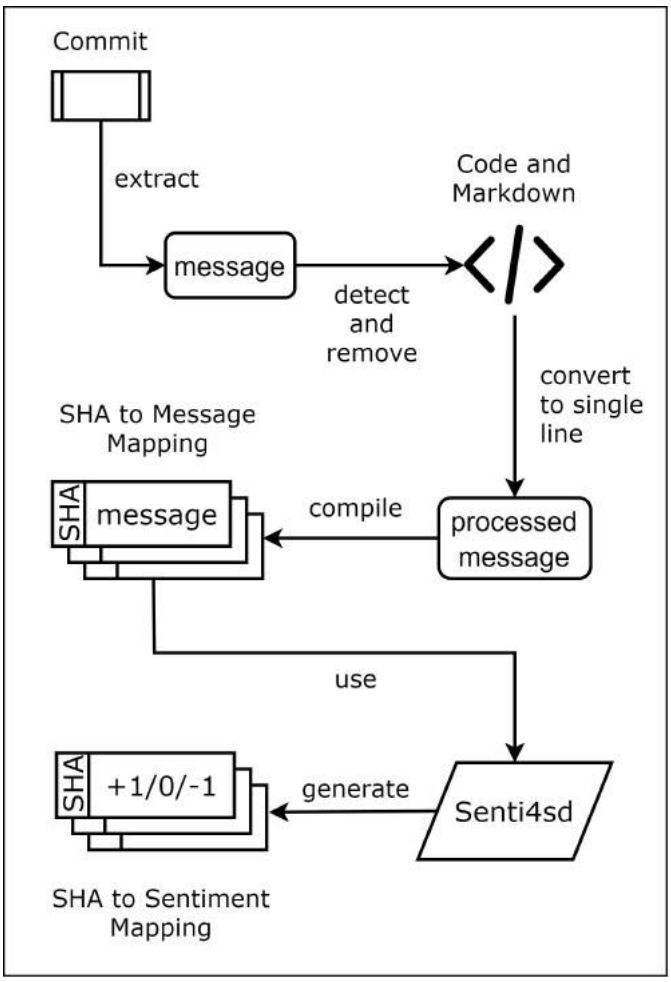

Figure 4.5: Method of extracting sentiment from Commit message

Senti4SD takes in texts as input and, after analysis, assigns a sentiment polarity label — negative, neutral or positive — to the text. Before this analysis, the Commit message is preprocessed to:

- detect and remove code components

- detect and remove markdown imposed by GitHub
- convert texts to a single line, by replacing newline characters with a full stop and a space

This is conducted for all Commits and finally a mapping between Commit SHAs and the messages are generated. This will later be used to map the sentiment score back to the original Commit. The text from the map is inserted into Senti4SD, which outputs the polarity labels. The labels are then transformed to their respective numeric representation — -1, 0 and +1. The numeric transformation helps in statistically analyzing the data. Lastly, using the map previously constructed, each Commit is assigned a polarity label and value. This process is shown graphically in Figure 4.5.

Table 4.1: Examples of Sentiment Values of Commit Messages from the Guava Project

Sha	Commit message	Senti4SD result	Value
bbab2ce	ARGHGH, guess I was in the wrong directory when submitting... amateurs...	Negative	-1
f5ad01f	Some fixes to java5-compatible compilation	Neutral	0
2346903	Almost got it right, but luckily Colin was here to help me.	Positive	1

Table 4.1 shows sentiment values after applying this process. The Commit messages are extracted from Google's Guava project. The table shows the three types of sentiment labels that can be extracted from Commit messages.

4.2.3 Dataset Description

The processes described previously are conducted on GitHub projects, since these software projects adopt Git, the version control system imperative for executing the Commit based functionalities of this study. Thirteen GitHub repositories are chosen based on popularity [102] and inclusion in the GHTorrent dataset [103].

Table 4.2: Repository Description

Project Name	**Lifetime (Years)**	**Releases**	**Contributors**	**Commit Number**
Guava	9	88	214	5134
Mockito	12	470	170	5190
Tomcat	5	210	31	21570
Commons-lang	16	87	127	5594
Hadoop	5	326	239	23271
Elasticsearch	9	262	1312	49416
Spring-Framework	16	165	416	20017
Selenium	8	119	466	24548
Netty	10	210	414	9630
Bukkit	9	67	106	1509
Clojure	11	148	144	3290
Facebook-Android-Sdk	7	106	76	1322
Actionbarsherlock	9	32	53	1480
Average	9.7	148	170	13228.54

As seen in Table 4.2, the repositories are all mature with an average 9.7 years of project life and 148 releases. Furthermore, with an average of 170 contributors, the projects contain the collaboration necessary to analyze inter-developer communication. Due to being open source, the projects enforce communication via Commit messages as their teams are geographically dispersed.

Extracting the Commits from the thirteen repositories yields the primary dataset consisting of a total of 171,971 Commits. Before sentiment analysis and conducting a study of correlation, the Commits are categorized according to the *Commit Categorization* section. The process produces four sets of Commits — FIC, pFIC, FC and FIF.

Figure 4.6, graphically shows the proportional existence of FIC and FCs, and their intersection, FIFs. The data shows that there is a mostly constant pattern in the proportion of the three Commits. FICs and FCs stand at nearly equal amount, FCs dominating by a small portion. For the projects Selenium and ActionBarSherlock, there is an exceptionally low amount of FICs compared to FCs. This can be attributed to multiple FCs tackling a common and recurring bug. On an average, 10% of the Commits are FICs, Commits that introduced bugs. 12%

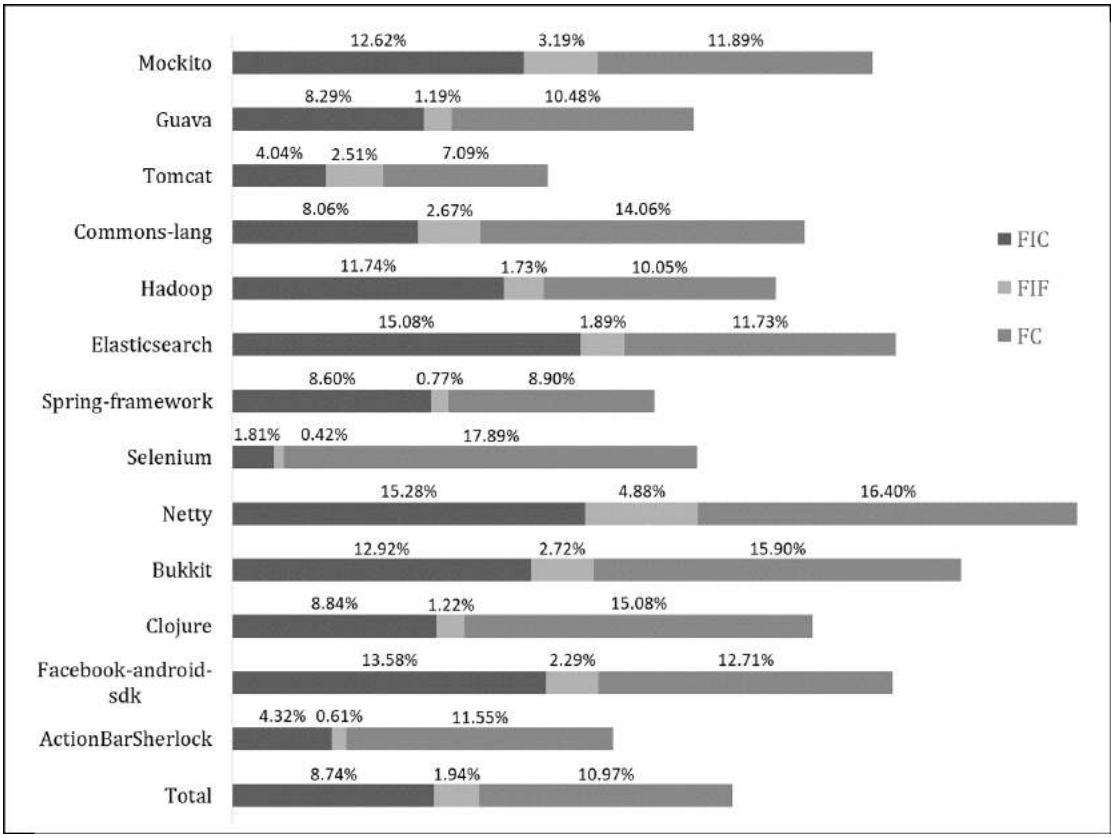

Figure 4.6: Ratio of FIC, FIF and FCs in projects from the dataset

Commits are changes that are related to fixing those bugs. The intersection of these two sets are the FIFs: an average of 2% Commits. Among the set of bug fixes, approximately 17% are incorrect fixes, as derived from the FIF to FC ratio.

4.2.4 Statistical Analysis

Using the different categories of Commits extracted, the sentiment in their messages are calculated. Using sets of special Commits — the four types — against their individual counterparts — regular Commits — statistical analyses are conducted. The following two methods are used for the statistical analysis:

- **Wilcoxon rank sum test** is conducted on the resulting sentiment data. The test determines whether the difference of means between two ordinal or interval non-parametric distributions is significant. It generates a p-value which, if < 0.05, rejects the null hypothesis: the means are not significantly

different. In Table 4.3, the reported p-values are modified with Bonferroni corrections to eliminate family-wise error rate.

- Additionally, to understand whether the proportion of different sentiment has any relation with the four types of Commits, a **Chi-square independence test** is conducted. The test shows whether observed frequencies of the two categories of variables have a significant association. In Table 4.4, the italicized values represent significance based on this test.

4.3 Study Result

The individual sentiment data based on the Commit categories are statistically analyzed to infer relation. Findings of the four Sub-Research Questions (SRQs) are described as follows.

Table 4.3: Commit Relation with Sentiment

		All Emotion		**Polar Emotion**	
SRQ	**Commit Type**	**Mean**	**p-value**	**Mean**	**p-value**
1	FIC	-0.085	$4.72e^{-16}$	-0.403	0.99
	Regular	-0.064		-0.402	
2	pFIC	-0.082	$3.27e^{-12}$	-0.436	$4.26e^{-07}$
	Regular	-0.063		-0.394	
3	FC	-0.119	$< 8.8e^{-16}$	-0.595	$< 8.8e^{-16}$
	Regular	-0.058		-0.366	
4	FIF	-0.016	$< 8.8e^{-16}$	-0.580	$< 8.8e^{-16}$
	Regular	-0.076		-0.397	
5	FIF	-0.137	$2.08e^{-05}$	-0.576	0.8565
	non-FIF FC	-0.106		-0.581	

SRQ1: How does developer sentiment relate to Fix-inducing Changes?

Observation: The first SRQ observes the relation between developer sentiment and Fix-inducing Changes (FICs). From Table 4.3 it can be seen that, when all emotions are considered, there is a significant difference of sentiment values between FICs and regular Commits. Based on the statistical evidence, it can be said that FICs are more negative than other Commits.

Table 4.4: Sentiment Proportions

		Polarity		Subjectivity	
SRQ	**Commit Type**	**Negative**	**Positive**	**Emotion**	**Neutral**
1	FIC	70.13%	29.87%	*21.01%*	*78.99%*
	Regular	70.12%	29.88%	*15.82%*	*84.18%*
2	pFIC	*71.78%*	*28.22%*	*18.92%*	*81.08%*
	Regular	*69.72%*	*30.28%*	*16.00%*	*84.00%*
3	FC	*79.77%*	*20.23%*	*20.03%*	*79.97%*
	Regular	*68.31%*	*31.69%*	*15.84%*	*84.16%*
4	FIF	*78.98%*	*21.02%*	*2.84%*	*97.16%*
	Regular	*69.86%*	*30.14%*	*1.76%*	*98.24%*
5	FIF	78.79%	21.21%	*23.83%*	*76.17%*
	non-FIF FC	79.07%	20.93%	*18.29%*	*81.71%*

Furthermore. from Table 4.4, it can be seen that both FICs and regular Commits have similar ratios of negative and positive emotions. Here negative sentiment outpopulates positive ones by at least 40%. However, FICs have more emotional messages than regular ones. FICs contain 6% more non-neutral messages.

Inference: Commits that introduce bugs are more negative and contain less neutral message than regular Commits. This information can be used for precautionary measures with sensitive changes. For instance, Commits with negative messages can be given better emphasis during reviews. More priority should be given on monitoring and maintaining the sentiment of developers, as that relates to problematic performance.

SRQ2: How does developer sentiment relate to the parent of Fix-inducing Changes?

Observation: From Table 4.3, it is observed that, parent of FICs are more negative compared to regular Commits, when considering both polar and neutral emotions. Table 4.4 shows that pFICs contain 2% more negative messages than regular Commits, while negative sentiment outnumbers positive ones by at least 39%. pFICs have 3% more emotional messages than regular Commits.

Inference: Commits prior to introducing bugs are found to be more negative and less neutral compared to regular Commits. This information depicts the

working situation prior to an error, as well as the state of the message left by the previous developer. Neutral messages should be posted so that it does not influence the next developer to make a mistake.

SRQ3: How does developer sentiment relate to Fixing Changes?

Observation: Results from Table 4.3 shows that considering all and polar emotions, FCs and regular Commits have a significant difference of sentiment values. Statistically, FCs are more negative than regular Commits. Furthermore. from Table 4.4, it can be observed that FCs have more negative (by 10%) and emotional (by 5%) than other Commits.

Inference: Commits that fix bugs are more negative and less neutral than regular Commits. This represents the emotional state of developers when tasked with solving bugs, proving the affect of complex assignments on developer emotion.

SRQ4: How does developer sentiment relate to Fix-inducing Fixes?

Observation: This SRQ is observed in two ways. First, FIFs are compared with regular Commits, showing a statistical significance in their negativity, as seen in Table 4.3. Table 4.4 shows that there are 9% more negative messages in FIFs than in regular Commits.

In the second observation, comparing incorrect (FIF) and correct fixes (non-FIF FC), Commits that correctly fix bugs are significantly less negative, with 5% more neutral messages.

Inference: Incorrect bug fixes can be differentiated from regular Commits and correct fixes using developer sentiment. The finding indicates that developers assigned with bug fixes will mistakenly add a new bug with a negative emotional state. Additionally, FCs that show emotional messages should be revised for possible bugs.

These findings provide insight into the developers' emotional patterns regarding software bugs and how it can be used to anticipate bugs being introduced.

4.4 Threats to Validity

This section presents potential aspects which may threat the validity of the study:

- **Threats to external validity**: The generalizability of the study's results is a major factor for the external validity. The analysis of this study has been conducted on thirteen projects from GitHub whose main language is Java. Concern can be placed on the use of a single main language. However, the collaborative artifact used for this study — Commits — are a constant regardless of the language used. Furthermore, the text written in the Commit messages are affected by the language to the extent of code snippets used as references. In this analysis, code snippets have been detected and removed as a preprocessing step to eliminate any such bias.

- **Threats to internal validity**: The correctness of the study's methods and procedures are the factors affecting the internal validity. Firstly, in the study, it is assumed that Commits related to bug fixing activities can be determined through their messages. This assumption, along with the use of search terms "bug", "fix" and "patch" are based on a previous study by Kim et al. [28].

  The second threat lies in the extraction of sentiment scores and their correctness. The study uses Senti4SD, which showed an accuracy of 79%, increasing in individual accuracies for each of the three values — negative, neutral and positive — from the baseline tool SentiStrength [85]. However, as mentioned by Calefato et al., their process of evaluation, similar to other sentiment analysis studies, depends on manual annotation. The annotations are open to evaluator bias as the perception of sentiment is a subjective matter.

4.5 Summary

This study has found the relation between developer sentiment and software bugs. Four types of Commits: Fix-Inducing Changes (FIC), parent of FICs (pFIC), Fixing Changes (FC) and Fix-inducing Fixes (FIF) have been categorized and their messages analyzed. Sentiment polarity values of the messages have been extracted using the classifier Senti4SD. The Commits are observed to contain 6% less neutrality than regular Commits. Additionally, there are more negative messages in these Commits than positive. On average, bug related Commits are significantly more negative than regular Commits.

This result differentiates Commits that are related to bugs from regular ones based on developer sentiment. The finding strengthens the patterns observed in previous studies and prompts further investigation into the quantifiable impact of emotions on software systems.

Chapter 5

Sentiment in Development Tasks and FIC

Inter-contributor discussion and reviews are integral aspects of the software development process [22]. These discussions shape the outcome of the software product, in both high and low level perspectives. However, these discussions, conducted in online collaborative artifacts, are subjective and open to interpretation. Moreover, contributors discussing on major problems and debating about the most efficient process of solving those problems can convey sentimental messages in the comments. Such emotional messages can influence the assigned developer, causing unintended or biased changes in the final code.

Sentiment analysis [1] can extract the value of such sentimental comments, and when conducted to analyze online collaborative artifacts, can derive effects of contributor sentiment. The previous chapter adopted this hypothesis and showed the relation between sentiment and developer contribution. It used the messages associated with Commits to extract quantifiable emotional value. Later that sentiment was utilized to differentiate between contributions buggy and otherwise. The findings demonstrate a significant relationship between contributions related to bugs and the sentiment in their messages.

This chapter extends these findings by incorporating contributor discussion and reviews on submitted code, which are likely proponents of affecting change. These, along with the code contribution, encapsulate a problem solving task in the software development process cycle. To integrate all these contents, Pull Requests [26] are used. Pull Requests in GitHub contain Comments, Reviews and Commits, all of which encompass a development task. To further understand how developer sentiment is related to bugs, Pull Requests that cause bugs are differentiated and the sentiment of their contents is analyzed. By analyzing the difference of sentiment between regular and buggy tasks, the effect of developer sentiment is derived. The experimentation of this chapter extracts Pull Requests of 6 well known GitHub repositories, which contain both code and contributor discussion. Sentiment is calculated using a tool specializing in the software engineering domain: SentiStrength-SE [84]. Next, or Fix-Inducing Changes (FIC)s [11] are detected from Commits which are used to categorize the Pull Requests. The statistical result shows that FICs, compared to regular Commits, contain more positive Comments and Reviews. Commits that precede an FIC have more negative messages. Similarly, all the Pull Request artifacts combined are more negative for FICs than regular Commits.

5.1 Introduction

Sentiment analysis — the process of extracting emotions from natural text [1] — has garnered significant interest in the field of software engineering, relating emotions with development activities. This surge of interest is due to the conceptual and practical expansion of sentiment analysis in the workforce. Emotions have been observed to affect not only personal lives, but professional ones as well. Employees are observed to show varying emotional expression in hierarchical communication and their sentiment can be related to high and low spikes of

productivity [20]. Understanding the extent of how employees' productivity differs based on their affective or emotional response has potential to change the perception of workplace culture [104]. Studies of this discipline solidified the effectiveness of sentiment as a tool to understand and improve upon the workers' performance and productivity. This also further motivated its adoption in the field of software engineering.

In the domain of software engineering, consideration of participant sentiment in development processes has seen multiple early stage studies. The existence of emotion has been observed in contributor submissions to extract patterns [5, 6, 4]. These patterns show fluctuation of developer sentiment in accordance with their development activities. However, direct impact of those patterns has yet been observed. The next step, therefore, lies in understanding how emotion relates to the performance of software engineers.

Software development processes differ from general organizational operations in the full adaptation and utilization of the online space. Online version control systems have become a key aspect of almost every software projects. GitHub is one of the most popular online version control and project management systems [22]. Its management aspect is built on the inclusion of collaborative artifacts such as Commits, Issues, Pull Requests, Comments, Reviews etc. These artifacts are directly linked to the source code of the software and are populated by contributor discussion. Since GitHub also contains all versions of the source code, its progression based on contributor discussions can be tracked using the artifacts.

Among these, Pull Requests are not only directly linked to a developer's series of codes but also contain discussions from other contributors. A Pull Request encapsulates its context to a single task — fixing a bug or adding a feature. Hence, analysis on a Pull Request can help obtain an understanding on how the discussions regarding a task can influence its outcome.

Among the many features and artifacts GitHub provides for project manage-

ment, this study is concerned with Pull Requests. Pull Requests are not only directly linked to a developer's series of codes but also enable insight from other contributors. These contain code submissions by contributors, to be reviewed and merged into the system. There are three main contents of a Pull Request, as visualized in Figure 5.1:

Figure 5.1: Components of a Pull Request

1. Comments: Contributors can post comments on a Pull Request's feed to describe issues or suggestions regarding the changes. These discussions influence the contributor's consequent changes or Commits.

2. Commits: A Commit is a contributor's documented modification to the source code. Each Commit is associated with a Commit message where the contributor describes the changes made.

3. Review Comments: Reviewers can leave comments directly on a changed code. While Comments can be descriptive and usually elaborate on the task

as a whole, Review Comments are used for commenting on a specific line of code. Newer Commits followed by Review Comments usually contain changes directly based on those Review Comments[1].

Pull Requests originate from an Issue, which is posted by contributors of the software, including clients, stakeholders, managers, moderators and developers. Multiple Pull Requests can exist under a single Issue, each solving the mentioned problem individually. Once opened, the Pull Request opens up a thread for further discussion about the topic at hand. This time, the discussion center around specific procedures of problem resolution. Based on these, an assigned developer submits new code. Assigned reviewers then critique the code on an even more lower perspective. According to the reviews, newer code is posted. The cycle repeats until either the reviews claim the solution too unfitting for further improvement, thus closing the Pull Request, or the solution sufficient and resolving the Issue. The process flow of a Pull Request is demonstrated in Figure 5.2.

Pull Requests not only contain all the changes made regarding a specific task by a contributor, but also all the discussion and conversations related to those changes. The motivation behind each changed code is retained in the Comments and Reviews. Since Pull Requests construct a direct link between Commits and the Comments and Reviews, it is a useful feature to extract a correlation between discussion and code changes.

This study extracts the emotions pertaining in a development task by individually calculating sentiment in the three mentioned contents: Comments, Commits (messages) and Reviews. After extracting sentimental values, these are correlated to Fix-Inducing Changes [11] — code that accidentally introduces bugs to the system, inducing its fix in the future. FICs are used for discovering and analyzing the introduction of bugs to a software. These can be detected from developer Commits [28], a component of Pull Requests, that document the changes a devel-

[1]this chapter refers to "Review Comments" as simply "Reviews"

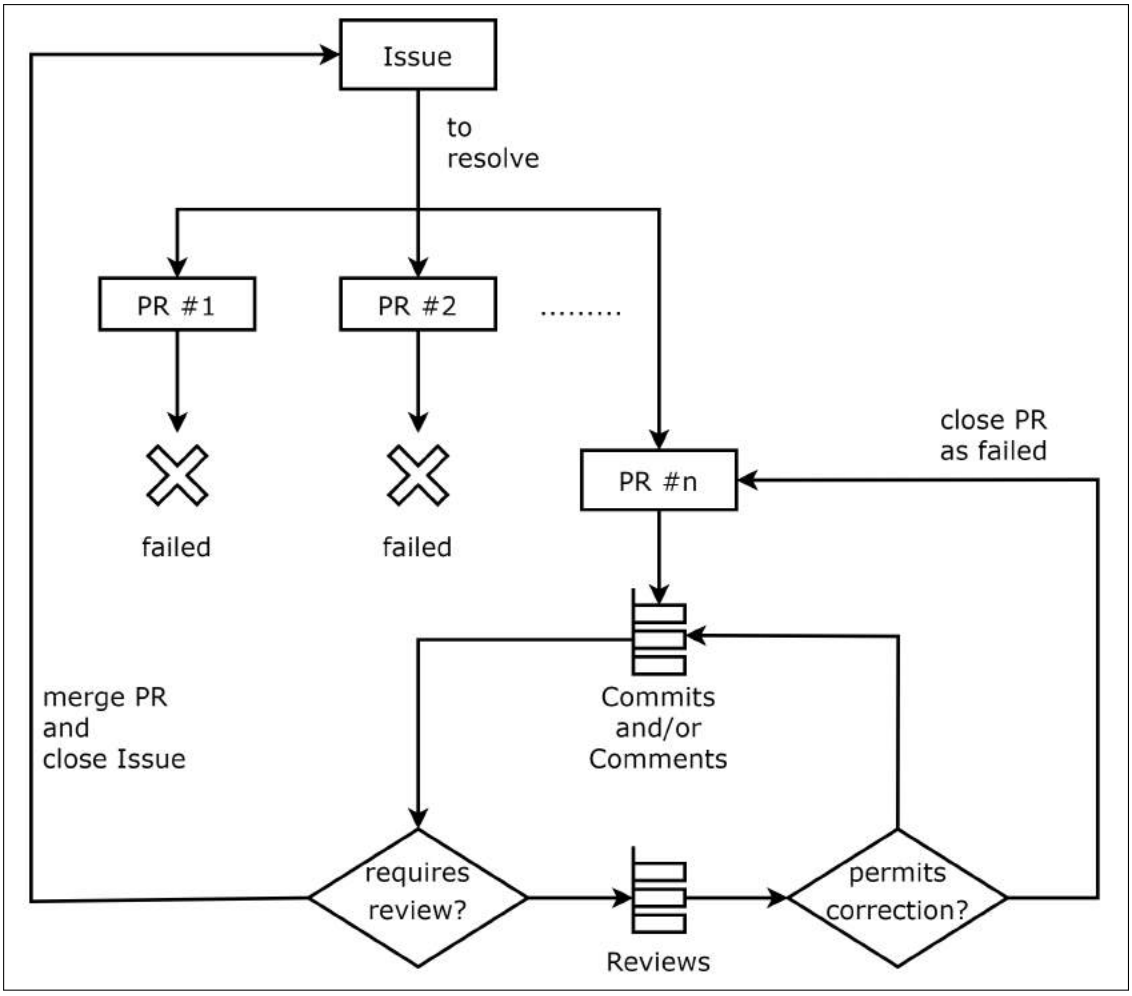

Figure 5.2: Process flow of a Pull Request

oper makes to the system. The introduction of bugs can be regarded as a metric for quantifying contributor performance. Therefore, observing whether sentiment in Pull Request components is related to FICs can yield an understanding of the effect of emotions on developer performance.

The study answers 4 Sub-Research Questions (SRQ) which are divided based on the utilization of different artifacts. All the SRQs aim to observe whether sentiment is related to FICs.

SRQ1: How does sentiment of Commits in Pull Requests relate to Fix-Inducing Changes? Pull Requests contain multiple Commits from a contributor to complete a task. Previous Commits can contain indication of the outcome of a certain change. Emotion in Commits can be extracted from its

message, in which the developers describe their changes.

Although the relationship explored in previous chapter also worked with Commits, in this chapter, the effect of the message's sentiment is correlated with the outcome of Commits that are posted afterwards. In other words, the previous correlation was linked between a Commit and its own message, while this one links a Commit and the message of the Commits that were posted before it in the Pull Request thread.

This new form of correlation is intended to estimate a buggy Commit before it is posted, whereas the previous one can only estimate both Commit and its associated message are posted. This SRQ will derive whether a developer's sentiment pattern in their contributions can contain indication on their future work.

SRQ2: How does sentiment of Comments in Pull Requests relate to Fix-Inducing Changes? Comments contain the discussion among contributors about the task at hand. These discussions start as an extension of those that were conducted in the parent Issue. In Pull Request, those are more granular, focusing on a single solution instead of inspecting all possible paths. Later on, the conversation progresses to include recommendations and criticism on the code from a high level perspective. Usually moderators, the assigned developers and optionally, other contributors participate in the conversation. These suggestions are put to practice by the coder in their subsequent Commits. Therefore, it can influence future changes.

This SRQ aims to use the series of Comments posted before a Commit to estimate that Commit's outcome. This is done by correlating those Comments with the likelihood of a future Commit being an FIC or otherwise. The findings will bolster the sentimental significance of contributor discussion.

SRQ3: How does sentiment of Reviews in Pull Requests relate to Fix-Inducing Changes? Reviews or Review Comments, as the name suggests, are subsequent comments to a change of code by a developer. In the Pull Request

thread, Reviews are posted as attachments to lines of code in a new Commit. These work as a suggestion towards the developer for further modifications. These are more direct than the Comments. New Commits are then posted based on these Reviews. Hence, Reviews can directly affect the changes that follow it.

This SRQ investigates the level of influence the sentiment in a Review has on the changes that follow it. A correlation further broaden the importance of code reviews in the software development process and help establish principles in how to properly conduct reviews without negatively affecting the code.

SRQ4: How does sentiment of all components in Pull Requests relate to Fix-Inducing Changes? While the previous SRQs explore the individual effects of the three major Pull Request contents, this SRQ aims to merge them together. All the three contents make up the thread's general disposition. While the Comments contain conversation about problem solving from a high level perspective, the Reviews tend to critique code-related inconsistencies. Lastly, the messages in the Commits, that are submitted as a result of the prior two contents, convey how the developer has perceived external input. Together their sentiment project a development task's overall emotional state. This SRQ, therefore, merges the three components to conclude the aim of the RQ, which aims to correlate the sentiment in development tasks with developer performance.

These SRQs are experimented upon in the following steps. First, taking six repositories from GitHub, all merged Pull Requests and Commits are extracted and stored. The contents of the Pull Requests are classified and divided based on the SRQs. The contents are assigned sentiment scores, ranging from -4 to +4, using SentiStrength-SE [84]. Each Commit is subsequently given a score by averaging the scores of prior contents. Commits are then categorized as FIC or regular, after computationally tracking changes that induce fixes. The two categories of Commits, with their separate scores, are statistically analyzed to derive correlation between developer emotion and FICs.

The result yielded from this exploratory research show that relations between emotion and FICs exist for all the four SRQs.

- Sentiment of Commits before an FIC is more negative than that of a regular Commit.
- FICs contain more positive Comments than regular ones.
- FICs are generated from more positive Reviews than regular Commits.
- Overall, FICs exist in Pull Requests more negative than regular ones.

5.2 Study Design

This study conducts sentiment analysis to find a correlation between Fix-Inducing Changes (FIC) and developer sentiment in project artifacts. Primarily, the methodology to conduct this research is divided into four parts. First, the artifacts are collected from GitHub repositories. Then, those are classified based on the four Research Questions mentioned above. Next, sentiment analysis is conducted on the artifacts using SentiStrength-SE. Lastly, FICs are detected from all the Commits of that repository which are used to categorize the Commits. These steps are then conducted on six different GitHub repositories. An overview of the methodology is provided in Fig 5.3 and descriptions in the following subsections.

5.2.1 Artifact Collection

Pull Requests are collected from the remote repository using the GitHub API [92]. Jcabi [105], a Java adapter for the API, is used. Jcabi provides helpful data containers for the different artifacts in GitHub as well as functions to extract their many properties. Using the API, all the 'closed' Pull Requests of a repository are traversed. 'Open' Pull Requests are excluded because these are still under

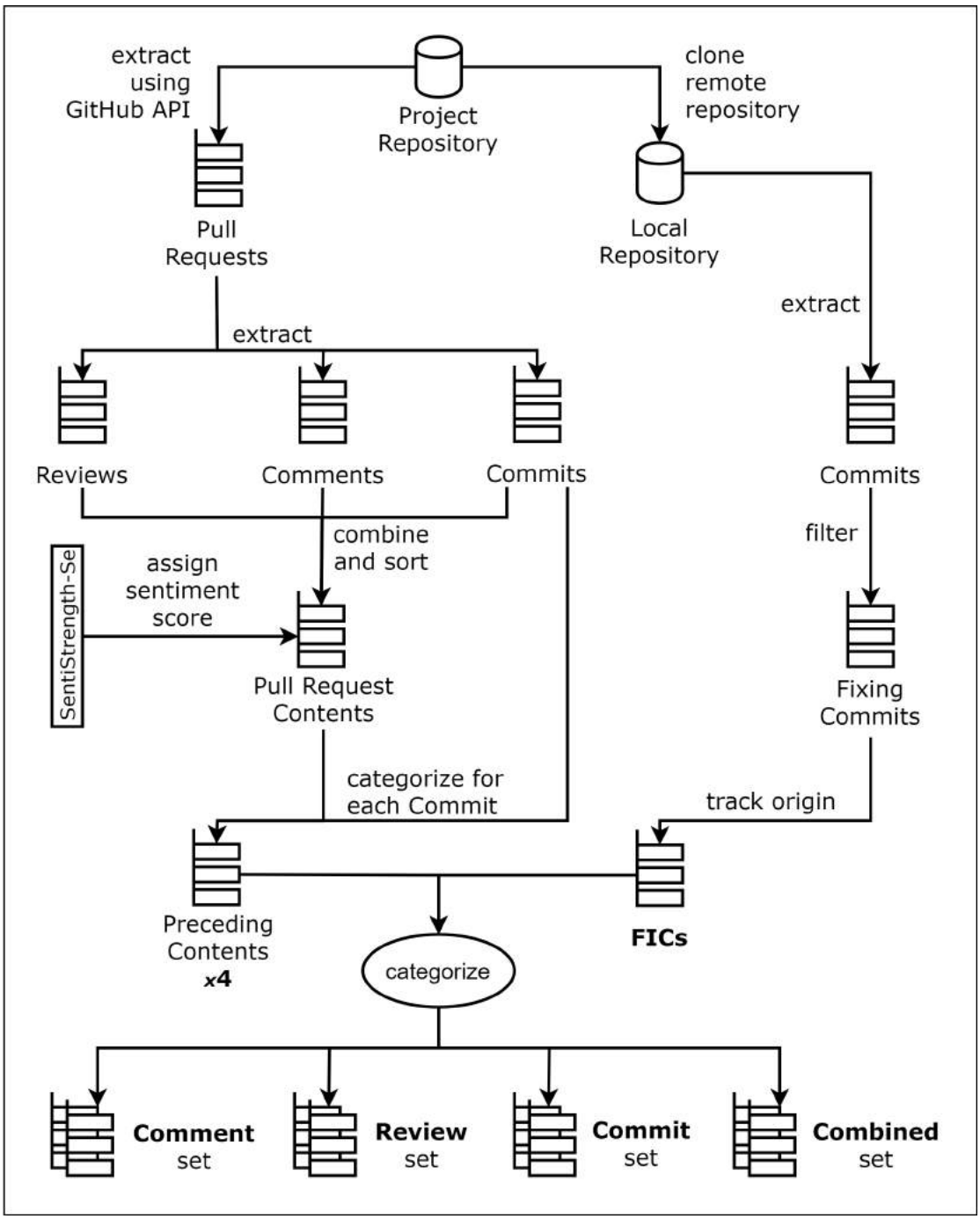

Figure 5.3: Diagram of the methodology

process. Among the 'closed' Pull Requests, only the merged ones are stored, because rejected ones do not contain code that exists in the system's base code. Each Pull Request links to three different contents: Commits, Comments and Reviews. The three contents are stored separately with sets named CM, CN and RV respectively. These are also combined under a common class, Pull Content. All four sets are sorted based on their creation dates. The collection process results in a set of sorted Pull Contents under every single Pull Request. This set can be denoted as, $PC = \{PC_1, PC_2, ..., PC_{n-1}, C_n\}$, where n is the total number of Pull Contents for that Pull Request.

Furthermore, all the Commits of the repository, regardless of their inclusion in Pull Requests, are fetched. The remote repository is cloned locally using Jgit [106]. All the Commits stored within the local repository are traversed and stored. The full list of Commits are necessary for tracking FICs.

5.2.2 Artifact Classification

After the collection process is completed, the contents are classified based on the requirements of the Research Questions (SRQ). Each SRQ requires its own uniquely processed set of sentiment values which will be used for differentiating between regular Commits and FICs. Each sentiment value (snt) belongs to a Commit and the unique processes differ based on the selection of Pull Contents that were posted prior to that Commit. Sentiment scores of the three types of Pull Contents are quantified through the $senti()$ function.

SRQ1

This SRQ tries to find out how a Commit in a Pull Request is affected by sentiment existing in all the previous Commits. So, for each Commit in a Pull Request, only the Commits that precede it are considered for sentiment analysis. The sentiment

score processed for this SRQ is based on equation 5.1.

$$snt_{PC_x} = \frac{\sum_{i=1}^{x-1} senti(PC_i)}{x-1}, \tag{5.1}$$

where $PC_x, PC_i \in CM$ and $1 < x \leq n$. The first Commit of a Pull Request is disregarded since no preceding Commit exists for calculating sentiment.

SRQ2

This SRQ tries to understand the emotional effect of Comments in a Pull Request on the outcome of a Commit. Usually a Comment contains discussion on the whole task instead of a specific part of code. Therefore, a Comment can influence any of the consequent Commits in a Pull Request. For this SRQ, all preceding Comments of a Commit are considered to calculate the sentiment score, as shown in equation 5.2.

$$snt_{PC_x} = \frac{\sum_{i=1}^{x-1} senti(PC_i)}{x-1}, \tag{5.2}$$

where $PC_x \in CM$, $PC_i \in CN$ and $1 \leq x \leq n$. Commits that have no preceding Comment are ignored from the final set of sentiment scores.

SRQ3

This SRQ experiments upon the correlation between Reviews and a Commit in a Pull Request. A Review contains instructions on how to rectify a specific code in a Commit. This prompts new Commits that change code according to the Review's instructions. The effect of a Review only spans in Commits that directly follow it. So, the sentiment score for a Commit (CM_x) is the average of all the Reviews that were posted since the previous Commit (CM_p). If no Reviews exist in this time-frame, CM_x is given the same score as CM_p since both are influenced by the same set of Reviews. Sentiment score of a Commit for both these cases is are

calculated based on equation 5.3.

$$snt_{PC_x} = \begin{cases} \frac{\sum_{i=p}^{x-1} senti(PC_i)}{x-p}, & \text{if } x-p \geq 1 \\ snt_{PC_p}, & \text{otherwise,} \end{cases} \tag{5.3}$$

where $PC_x, PC_p \in CM$, $PC_i \in RV$ and $1 \leq p \leq x \leq n$. Commits that have no preceding Review are discarded since no Review influences these.

SRQ4

The last SRQ takes all the Pull Contents into consideration. The general sentiment of a Pull Request can be observed through this. The sentiment score for this SRQ is as shown in equation 5.4.

$$snt_{PC_x} = \frac{\sum_{i=1}^{x-1} senti(PC_i)}{x-1}, \tag{5.4}$$

where $1 \leq x \leq n$. Commits with no preceding Pull Content are not considered.

5.2.3 Quantifying Sentiment

Sentiment polarity scores for the artifacts collected and classified previously are calculated using SentiStrength-SE. It is tool specialized for text in the context of software has been used as a state of the art tool for sentiment analysis studies in software engineering [101, 107].The tool first takes as input one or more sentences of text. The texts are then divided into individual sentences which are tokenized, filtered, stemmed and lemmatized. Scores for each of the tokens are assigned based on the sentiment dictionary. The dictionary consists of words or tokens and their corresponding sentiment polarity score from -5 to +5. The polarity indicates the type of sentiment — negative (< 0), positive (> 0) or neutral ($= 0$) — and the level of severity, based on the distance from 0.

Using the individual scores of each token, the sentence as a whole is assigned

two polarity scores — a negative and a positive point. The two points consist of the lowest negative (neg_{min}) and highest positive (pos_{max}) token scores in the sentence respectively. If no negative or positive token exist, the default polarity point is -1 and +1 respectively.

Subsequently, the complete text is assigned two polarity scores, taking the highest ones from the individual sentences. This also follows the default -1 and +1 scoring rule. The final score of the text is the sum of the two points,

$$senti(text) = neg_{min} + pos_{max} \tag{5.5}$$

The final score has a range of -4 to +4. -4 indicates a strong negative emotion and +4 a strong positive one. The severity decreases as the score nears 0 with 0 being a neutral emotion.

5.2.4 Fix-Inducing Change Detection

FICs are the erroneous changes to code that threaten the correctness of the system [11]. Tracking these changes provides insight into the causes of introducing bugs — both technological and behavioural. FICs are detected in the following steps:

1. For each Commit extracted using Jgit [106], its message is analyzed to check whether it is a 'Fixing Commit'. Fixing Commits are the changes to code that remove or fix bugs. Therefore, by identifying the codes that fixing Commit removed, the exact lines of code that possibly created the bug can be found. The Commits are filtered by the inclusion of these keywords in the message: "bug", "fix" and "patch" [28]. The filtered Commits are regarded to be related to bug fixing activities, hence are labeled as fixing Commits.

2. In the next step, modifications conducted in the fixing Commits are extracted. Modification of source code in a single Commit can be found by analyzing the difference between that Commit and its parent Commit. A

source code can consist of format change as well as code change. To find the changes introduced in a fixing Commit, all changed files are compared using DiffJ [108] tool. The tool compares changes between the two versions of that file existing in the fixing Commit and its parent Commit. DiffJ also disregards format changes, considering only source code modifications.

3. In the last step, the FIC is detected using the output of DiffJ tool[2]. The tool's output is the lines of code that were modified, which can be interpreted as errors that were mitigated. Finding the source of these lines can lead to the Commit that introduced errors, hence the FIC. The sources are tracked using Jgit's "blame" function, which finds the exact Commit where a line of code is added. This generates the list of FICs while the rest are considered regular Commits.

This process of detecting FICs is similar to [28]. Manual check for each repository shows that the Fixing Commits are indeed intended for bug fixes.

The end result of conducting all the four processes of the methodology are 4 paired sets of sentiment scores. Each paired set consists of one set of sentiment scores for FICs and another for regular Commits. The 4 pairs are based on the classification for each of the SRQs.

5.2.5 Repository Description

To conduct this research, six well known [109] Java projects are chosen from GitHub's repositories. Java is the language of choice because it provides necessary resources for extracting Fix-Inducing Changes (FIC) from the code: DiffJ. These repositories are of open source projects, which ensure that the artifacts are publicly available. These also house contributors who do not have a central workplace and need to communicate through online collaborative artifacts.

[2]DiffJ was used Instead of Jgit as it omits format and whitespace changes.

The six projects with their repository details are displayed in Table 5.1. The projects show maturity with an average project life of 7 years and average (median) 189 contributors. All the projects have a significant number of Commits to work with ranging from 4,798 to 23,550, covering projects of different sizes. With an average of 1,115 Pull Requests, the repositories together hold a sizeable number of contents to analyze on.

Table 5.1: Repository Statistics

Project Name	**Lifetime (Years)**	**Contributors**	**Commit Number**	**Pull Requests**
Google Guava	5	187	4798	469
Mockito	6.7	155	5019	826
Apache Commons-lang	10	115	5396	431
Apache Hadoop	4.8	191	21435	1073
Selenium	9.3	435	23550	1561
Spring Frame-work	6.4	378	18022	2331
Average	7	189	13036	1115

5.2.6 Content Description

Applying this study's methodology on the six projects produced 4 paired sets of sentiment scores. All pairs contain a set of FICs and a set of regular Commits. Fig 5.4 provides numeric comparison between the two types of Commits. The four sets represent the distinct scenarios of the four SRQs.

The Figure shows a large disparity in numbers between the two types of Commits. On an average only 6% Commits are FICs. This indicates the effectiveness of the review processes in Pull Requests, where each code is scrutinized methodically to remove any shortcoming.

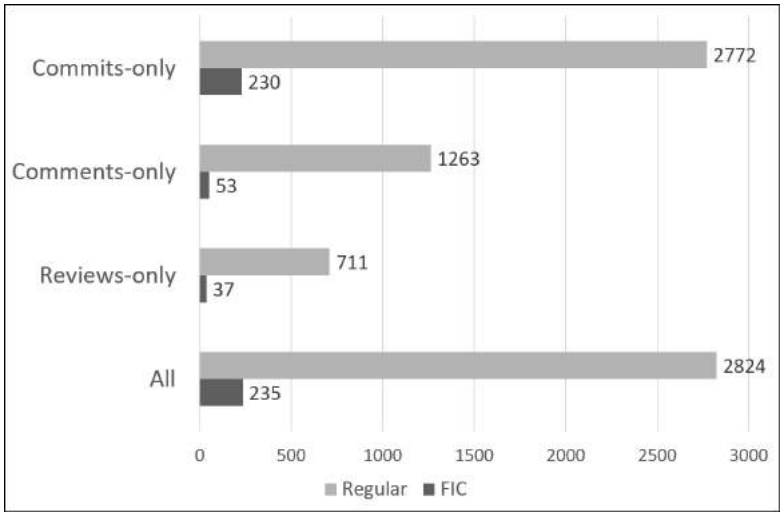

Figure 5.4: Comparison of FICs and regular Commits in different scenarios

5.3 Study Result

Each of the four paired sets are analyzed individually to answer the four Sub-Research Questions (SRQs). Analysis and discussion of each SRQ is as follows.

SRQ1: How does sentiment of Commits in Pull Requests relate to Fix-Inducing Changes?

For this SRQ, for every Commit in a Pull Request, all the previous Commits of that Pull Request are considered. Sentiment scores are calculated from the messages of those Commits. These are averaged and assigned to the Commit being analyzed. The process is repeated for all the Commits of all Pull Requests from all the projects.

Table 5.2 demonstrates the difference of sentiment for FICs and Regular Commits. It is observed that, on an average, the Commit messages before an FIC have less positive sentiment than a regular Commit's. A p-value less than 0.05 calculated from a Wilcoxon Rank Sum test tells that difference in the mean is statistically significant.

Furthermore, FICs contain more negative Commits than positive (31% to 18.3%). This difference is larger than that of regular Commits (16.3% to 12.2%). In terms of neutrality, Commits before FICs are 21% less neutral than regular

Table 5.2: Analysis Results

		Artifacts			
		Commits	Comments	Reviews	All
FIC	Neutral	50.00%	9.40%	29.70%	41.30%
	Negative	31.70%	3.80%	10.80%	28.50%
	Positive	18.30%	86.80%	59.50%	30.20%
	Mean	8.70E-05	0.37	0.11	0.03
Regular	Neutral	71.40%	26.20%	53.40%	48.90%
	Negative	16.30%	9.90%	15.30%	17.90%
	Positive	12.20%	63.90%	31.20%	33.20%
	Mean	0.004	0.26	0.09	0.04
p-value		0.019	0.031	0.022	0.04
Verdict		Effect Exists	Effect Exists	Effect Exists	Effect Exists

ones, meaning Commits before FICs show more emotion.

These results suggest that negative emotions in contributor Commits can indicate an FIC to come. If a contributor, when coding for a task, submits Commits that contain, in majority, negative sentiment, then their next Commits are prone to be FICs. This information can help in prioritizing or focusing reviews. Commits after a series of negative ones should be reviewed more intensely for possible introduction of bugs.

SRQ2: How does sentiment of Comments in Pull Requests relate to Fix-Inducing Changes?

Pull Requests contain Comments as a medium for discussion among multiple contributors and for this SRQ, the effect of these discussions is observed. FICs and regular Commits are given sentiment scores by averaging the scores of all the Comments that precede it. Comments consist of queries on the problem and solution, high level criticism of the code's approach, and suggestions on how to solve and improve. Their sentiment can alter developer emotion and thus affect their work. The results align with this assumption.

Table 5.2 shows that, based on the p-value from Wilcoxon Rank Sum test, Comments before FICs are significantly more positive than that of regular ones. Positive Comments outnumber both neutral and negative Comments by a large

margin. FICs have almost 23% more positive Comments posted than regular ones. Negativity for FICs is very low, ranging around 4%. Lastly, neutrality is more common for regular Commits, containing 16.8% more neutral Comments than FICs.

The results indicate that too much positive emotions in discussion may lead to buggy code. Positive emotion in Comments are caused by gratitude or praise. These are needed as positive reinforcement for the developer. But, according to the results, these can cause the opposite outcome. Positive sentiment can turn developers overconfident and careless, reducing their ability to scrutinize their own code. Furthermore, in case of multiple contributors reviewing a task, one contributor's positive sentiment can bias the others' judgement.

SRQ3: How does sentiment of Reviews in Pull Requests relate to Fix-Inducing Changes?

Reviews are an essential component in Pull Requests where reviewers leave comments linked to the code. These contain requests, suggestions and questions to the developer regarding specific lines of codes. Reviews, contextually, are nearer to the code than Comments. Hence, their area of effect is smaller and more immediate. New Commits abide by Reviews posted directly before them. The sentiment of Commits for this SRQ is calculated based on the nearest Reviews.

Statistical analysis shows that Reviews, like Comments, are more positive for FICs than regular Commits. As Table 5.2 demonstrates, based on the Wilcoxon Rank Sum test, the difference of sentiment is significant. FICs contain almost 30% more positively labeled Reviews than regular Commits. The ratio of positive Reviews is more than half of all sentiments (almost 60%). Although both types of Commits have almost similar percentage of negative Reviews (11% and 15%), neutrality is more dominant for regular Commits (53% to 30%).

This result is similar to that of SRQ2, but of lesser numerical extent. Reviews that provide positive reinforcement can also distort a coder's ability to be careful

with the code. The presence of emotion can confuse the coder in understanding the impact of their shortcomings. Reviews being directly linked to code, should be worded as formally as possible, omitting tangential discussions and being to the point.

SRQ4: How does sentiment of all components in Pull Requests relate to Fix-Inducing Changes?

All components of a Pull Request determine the general sentiment conveyed regarding a task. While Comments and Reviews show contributor reception to code, Commit messages hold contributor reaction to feedback. Together these set the tone of a virtual atmosphere, which in turn can influence the quality of code. A negative atmosphere has the potential to cause adverse effect in a developer's performance. The statistical analysis show that that is possible.

Results of the statistical analysis are provided in Table 5.2. It shows that, on an average, the Pull Requests of an FIC are more negative than that of regular Commits. Wilcoxon Rank Sum test solidifies the difference. The difference in ratio between the two categories of Commits is minimal. The internal ratios of the different sentiment polarities are also similar in both cases. Regardless, neutrality is higher for Pull Requests of regular Commits. This means that more emotion is displayed in Pull Requests that influence FICs.

The first three SRQs dealt with the components separately. All three observed that polar emotions play an adverse effect on the code. While Comments and Reviews are more positive for FICs, previous Commits are more negative. Comments hold the least amount of neutral sentiment, owing to emotional discussions. Reviews also show a similar pattern, but at a smaller quantity. Commits, as well, show that lack of neutrality results in FICs. The results of the last SRQ reinstates this effect of neutrality.

Despite FICs containing more negative artifacts on an average, the larger disparity between FICs and regular Commits are in the neutral artifacts. Similar

to individual effects, together the components are less neutral for FICs. This demonstrates a pattern where predominance of polar emotion in discussions lead to changes that introduce bugs.

Based on the inference of the statistical outcome, the information can be applied in the following two ways:

- **Anticipating FICs by analyzing sentiment:** The results show that FICs are preceded by: (1) negative Commits, (2) positive Comments and (3) positive Reviews. These can help in anticipating a buggy code. Moderators and reviewers can analyze the sentiment of the components of a Pull Request, and bolster reviewing processes if the mentioned patterns are found. Future studies can develop prediction models based on the individual and combined effects of the components.

- **Moderating the language of reviews:** Since emotions affect FICs, discussions in Pull Requests can be curated accordingly. The results warn of the detrimental effects of polar emotions. Neutrality was constantly larger for regular Commits compared to FICs. Furthermore, the opposite effects of Comments and Reviews, and Commits show that, a balance between positive and negative emotions is required. Based on these instructions, moderators and managers can administer how collaborative artifacts are used and how reviews should be worded.

5.4 Threats to Validity

The results of this study have been produced from Pull Requests of 6 Java projects from GitHub. All projects are live, adding, merging and closing Pull Requests on a regular basis. These can also be reopened for further activities. Hence, the dataset cannot be recreated by following the processes mentioned alone. Table 5.3 provides the date the repositories were last accessed and the latest Pull Request

extracted. A correct replica of the dataset can be created by filtering based on the dates mentioned and checking the ID of the highest Pull Request extracted.

Table 5.3: Project Last Access Information

Project	Last Access Date	ID of Latest PR
Google Guava	2-Jul-19	3519
Mockito	4-Jul-19	1727
Apache Commons-lang	3-Jul-19	433
Apache Hadoop	6-Jul-19	1049
Selenium	4-Jul-19	7347
Spring Framework	3-Jul-19	23191

For sentiment analysis, the dictionary provided by SentiStrength-SE [110] is used, but with a single modification. The dictionary gives "mess*" a sentiment score of -2, to reflect the negative connotations of words like "mess", "messy", "messed" etc. But the prefix also includes "message" which is a neutral word. Hence, the dictionary is updated for this study, replacing "mess*" with the individual variants of "mess".

5.5 Summary

This study observes whether developer sentiment in software development artifacts can indicate the introduction of bugs in the system. Fix-Inducing Changes (FIC) are used as the metric to determine the origin of bugs. Sentiment is derived from GitHub's Pull Requests, which contain collaborative and contributory artifacts like Commits, Comments and Reviews. SentiStrength-SE is used for calculating the sentiment polarity of the artifacts. FICs are detected through an automated approach of filtering Commits that fix bugs and finding the origin of the code these remove. Finally, Commits are categorized as FICs and regular Commits, and assigned sentiment scores based on preceding artifacts.

Analysis conducted on six Java projects shows that there are significant differences of sentiment scores between FICs and regular Commits. Comments and

Reviews before FICs have a more positive sentiment than that of regular ones. Conversely, Commits and the overall discussions preceding FICs are more negative than ones before regular Commits. This information suggests that whether bugs are being introduced in the system can be indicated through the emotions of collaborative artifacts that precede a change.

Future studies can utilize this applicability of sentiment for understanding developer performance. Other artifacts, like Issues can be considered, where the participation of clients and other contributors tend to be more opinionated.

Chapter 6

Sentiment in Contributor Discussion and FIC

In the software development process cycle, every change in the software is preceded by a series of discussion among the end users, stakeholders or contributors of the software. These discussions can encompass description about new problems with the software or new requests by the consumers, the viability of addressing these cases, and the mechanism with which these can be resolved [111]. All of these conversations results in a guideline for the assigned developer on the potential solution. Although the guideline is described from a high level perspective, it shapes the end result. These primary discussions can also adversely affect the software if the messages conveyed are perceived incorrectly [7, 9]. Inconsistent perception can be caused by ambiguity and subjectivity in the text, often associated with emotional messages. Sentiment analysis can unearth the inconsistencies in those messages based on the presence of human emotions in the written text.

This chapter uses this insight to expand on the findings of the previous ones. The last two chapters observed the sentiment in texts present in developer contributions and problem solving discussions respectively. It was seen that both are correlated with buggy code, or Fix-Inducing Changes (FIC) [11], which were used

to depict the instances of bugs being introduced to the system. In this chapter, the correlation includes conversations during problem origins, before tasks are initiated. This completes the consideration of all the different textual exchanges conducted from the introduction to a problem in a software to its resolution through developer code.

In this chapter, GitHub Issues are used for extracting conversational text. Issues are the primary entry point for concerns about the software [25]. In Issues, the title and body contains description about the concern while the following comments are used for discussing a way to resolve these. Since these comments are not strictly code-specific, subjectivity can easily and unintentionally be inserted, which can be detected through sentiment analysis. Pull Requests and change Commits origin from these Issues. These can be used for detecting buggy changes and correlating these back to the original Issue. Using SentiStrength-SE to extract sentiment, it is seen that Issues, in general are predominantly neutral. These are also more positive than Pulls. By tracing Pull Requests that contain or Commits that are FICs, a significant correlation between Issue sentiment and buggy changes is found. Issues resulting in FICs consist of more positive discussion than Issue resolutions with no negative outcomes.

6.1 Introduction

Modern software development processes depend on online collaborative artifacts for inter-contributor discussions. These discussions encompass emerging concerns related to the software and their solutions. Issues in GitHub are one such artifacts. Discussion in GitHub Issues contain requests and complaints from end users or consumers of the software product. Their initial post contains a title summarizing the concern, and a body or description that catalogues, in detail, the source, type and behaviour of said concern. This is followed by conversation among con-

tributors to solve the Issue. Contributors can include the consumer who posted the issue, the client in charge of the product, the admins and moderators of the project and developers whose responsibility it is to suggest or code a solution. The solution can be in the form of debugging existing code or adding new ones as features. Based on the conversations, the developers write the program. The code submitted by them prompts further discussion and consequent code modifications. The end results are Commits that solve the Issue, which, based on the process cycle mentioned, are influenced by the discussions that preceded it. Therefore, it is constructive to qualitatively analyze these discussions in order to understand the effects of contributor interaction on the adverse effects on the resulting code.

One way to analyze the quality of discussions is sentiment analysis. Sentiment Analysis is the extraction of human emotions from written natural text [1]. Since Issue Comments are written in natural language and with more informality than technicality, sentiment analysis will be able to extract the emotional values of the text corpus. These values, traced to the end result, can yield the correlation between the quality of discussion and the quality of code.

Sentiment analysis has previously been studied in the software engineering domain to extract patterns and relate sentiment to various project properties. Issues have been a popular artifact of choice in these analyses. Issues, from GitHub and Jira, have been analyzed for sentiment to understand emotional patterns [4] and apply it in predicting Issue resolution time [7, 8]. Other studies include the effects of sentiment on performance degradation [27] and contributor turnover [9]. Sentiment in GitHub Commits has been extracted [5, 6] and related to the day of activity, change size and personnel diversity. However, these studies do not relate the emotional patterns with the quality of the source code, which would provide a quantifiable impact of those patterns.

Quality of code can be derived from software Commits, which contain the changes made by a developer. For instance, whether a code is buggy can be

detected from Commits by finding Fix-Inducing Changes (FIC). FICs are Commits that introduce bugs to the system and therefore induce fixes [11]. FICs have been studied to understand the circumstances and project properties that lead to the introduction of bugs. For instance, FICs have been related to day of coding and change size [11], incorrect fixes [29], code smells and refactoring efforts [98], clone couples [30] and more. This study aims to relate FICs with sentiment, to understand how contributor discussion in GitHub Issues can result in bugs.

The relation between GitHub Issues and Commits is conceptually inter-dependent. Issues need to be resolved by changing the software's code, which is possible through submitting and merging Commits. Therefore, Commits are necessary for closing Issues. On the other hand, Commits are parts of a task assigned to a developer. These tasks are formed in Issues. Despite the conceptual dependency, in practical terms, Issues in GitHub are not directly connected to Commits. Rather, Pull Requests, or shortly Pulls, work as intermediaries between the two artifacts [26]. Although not all Commits are contained in Pulls, it is prevalent in open source projects. In these projects, developers work on their own versions of the main project, or Forks [112], which do not grant direct merge access. Therefore, the Commits are reviewed through Pulls. Once the Pulls are merged, the origin Issues are resolved. In this study, to understand the impact of discussions conducted in an Issue on the correctness of Commits originating from it, Pulls are integrated in the analysis.

The following two Sub-Research Questions (SRQs) will work as a guide to reach the goals of this study:

- **SRQ1: What is the general sentiment of the discussions in GitHub Issues?** Despite containing some similar contents as Pulls, Issues are used for a different purpose with a different set of contributors participating in the dialogue. The text in Pulls revolves around code-specific discourse, reviewing shortcomings and suggesting edits to match a satisfactory solution.

By contrast, Issues contain text that provide a high-level guideline to a solution. Whether these differences translate to the sentiment in their respective texts, and if so, how they differ is the intended findings of this SRQ.

Firstly, this SRQ observes the emotional patterns in GitHub Issues in terms of polarity and severity. Secondly, a comparative analysis is conducted between the emotional patterns of Issues and Pulls.

- **SRQ2: How is sentiment related to Fix-Inducing Changes (FICs)?** Once the emotional patterns are retrieved, these are traced to the resulting code. While the patterns provide insight into how contributors interact among themselves, the correlation will derive the impact of the interactions.

  This SRQ analyzes Issues and subsequently the Pulls that originate from these, extract their sentiment and statistically differentiate between ones that lead to FICs and ones that do not. In doing so, it is observed whether discussions that end up introducing bugs to the system contain distinguishable sentiment along with the nature of the extracted sentiment.

- **SRQ3: What are the comparative effects of Issues, Pulls and Commits in the relation between FICs and sentiment?** So far, individual relations between sentiment and performance have been measured using three different artifacts. The conversation that encompasses Issues also include Pulls and Commits. Therefore, from Issues, sentiment from all the other artifacts can be derived. This provides the opportunity to compare how these different artifacts affect the relation between sentiment and performance individually and together.

  This SRQ extracts sentiment from different contents of the major artifacts: Issues, Pulls and Commits. The sentiment is related to FICs and the effect sizes of these relations are measured. The measurements are compared to observe the varying levels of effects the artifacts pose on the correlation.

The methodology of this study concerns three aspects: creating relations among the artifacts of GitHub, analyzing the sentiment of those artifacts and detecting FICs from Commits. The link between the artifacts in GitHub — Issue, Pull and Commit — are derived based on references and correlation. The sentiment of the description and comments of the artifacts is calculated using SentiStrength-SE [84], a sentiment analysis tool specialized for the software engineering domain. The distribution of sentiment is observed based on the resulting scores, that range from -4 to +4, with 0 as the neutral emotion. To detect FICs, Commit messages are first parsed to list changes that fix bugs. These are then tracked back to extract changes that introduced those bugs, hence FICs. Statistical analysis is then conducted on the resulting sets of sentiment values to observe the relation between sentiment and FICs.

The findings show that almost 59% of contents in Issues are neutral, while positivity is more prevalent in emotional messages. Sentiment in Issues are less neutral than that of Pull Requests. Titles in Issues are more formal and neutral than their bodies and comments. Contents of Issues that lead to bugs are more positive than ones that contain no FICs. Lastly, it is observed that different contents from the artifacts contain differing effect sizes for these correlations; contents from Pulls and Commits containing the larger ranks. These findings solidify the significant relation between contributor sentiment and buggy code.

6.2 Study Design

The methodology of this study contains three principle aspects. First, to connect Issues with Commits, a graph of the artifacts is generated. Sentiment analysis of the textual content contained in the artifacts is then conducted. From the Commits extracted, Fix-Inducing Changes (FICs) are tracked and, using the graph, connected to Issues. This design is conducted on 50 GitHub projects. From the

generated data each Sub-Research Question (SRQ) analyzes information differently using individual statistical approaches. These steps are described in the following sections.

6.2.1 GitHub Graph Generation

This study's primary goal is to understand whether the discussion in Issues holds any influence on the resulting code. This requires a link between the artifacts. Through this link, correlations can be established.

As shown in Figure 6.1, software development in GitHub flows through Issues and Pull Requests (or, Pulls). Issues are posted based on the outcome of Commits or releases and contributors discuss its aspects and solutions in the comments. Based on the discussion, developers open new branches as Pulls and add their Commits to fix the Issue. Their Commits in the Pull prompt reviews from moderators and reviewers. The Pull is merged if the code is satisfactory (Pull 2), which also merges the associated Commits to the master branch. Otherwise the Pull is closed (Pull 1) and its Commits are rejected. However, not all Commits are necessarily posted through Pulls and can be directly added to the master branch.

As Figure 6.1 shows, there are no direct links from Issues to Pulls and Commits. Since this study requires a relation between Issues and Commits to understand the impact of contributor discussion on developer code, the links are rather derived. To link Issues to the Pulls and Commits, references are extracted from their text — Pull title and body, and Commit message. These references are manually written by contributors to indicate the parent Issue. References can be written in any of the following formats [76], where *NN* is the Issue number:

1. https://github.com/owner/repository/NN,
2. #NN,
3. GH-NN and

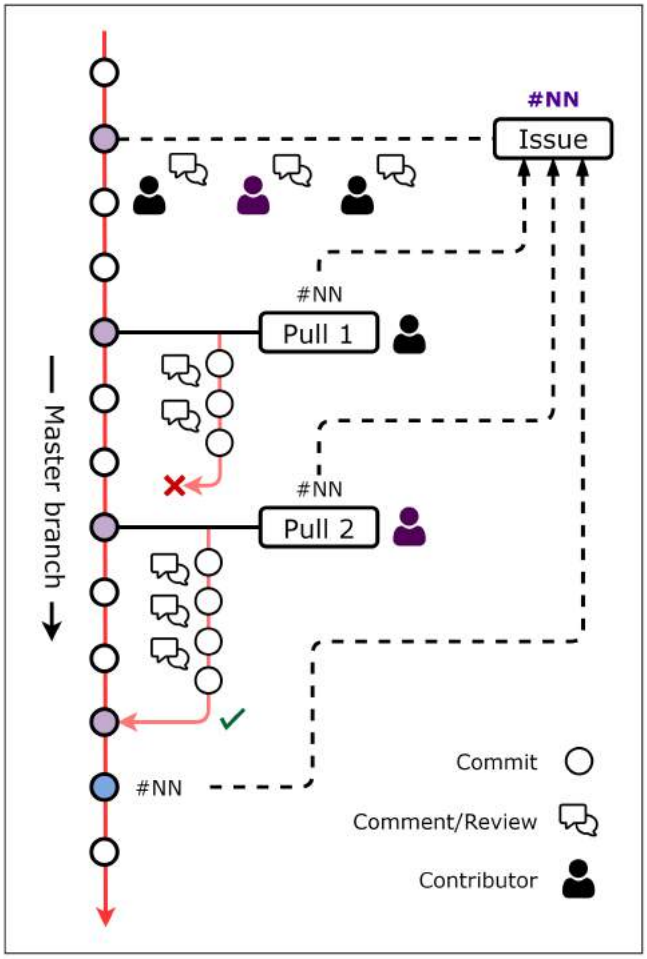

Figure 6.1: Flow of activities with GitHub artifacts

4. owner/repository#NN

Linking Issues using references creates three types of relations between Commits and Issues. Figure 6.2 shows an example artifact graph generated from the project Mockito. Here, Issues contain Commits in three paths: one immediate and two derived. Via the immediate path, it is seen that from a total of 746 Issues, 196 are referenced from the 5491 Commit messages. The two derived paths are through merged and rejected Pulls. From these relations, the rejected path contains 54 references. But these are discarded from further analysis since their Commits do not exist in the master branch and therefore cannot affect the software. Instead, the merged Pulls with a total of 574 references are adopted for analysis, along with the Commits extracted from the direct path.

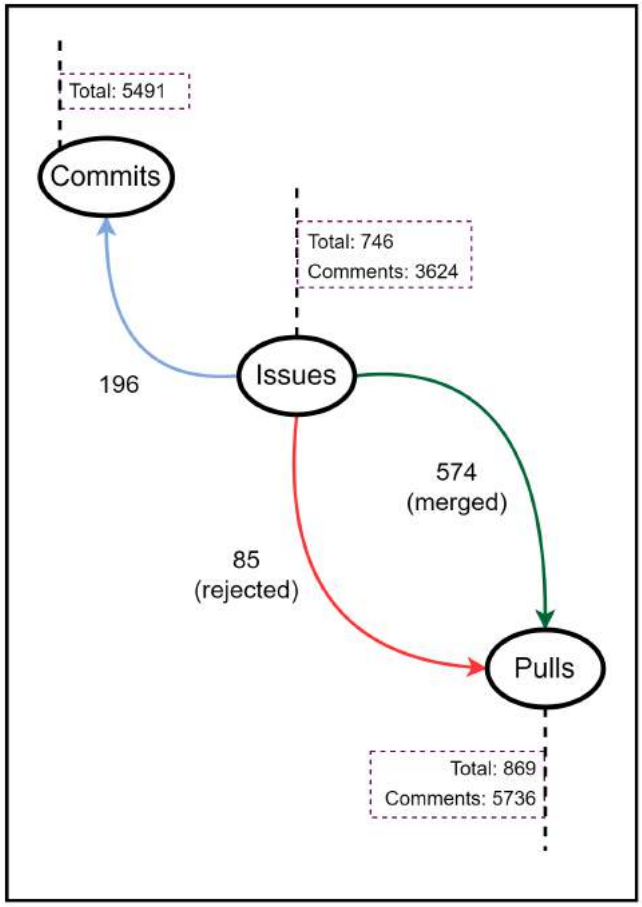

Figure 6.2: Extracted links between artifacts of Mockito

6.2.2 Sentiment Analysis

To qualitatively analyze the discussion in the extracted artifacts, sentiment of their text is calculated. For this study, SentiStrength-SE [84] is used for extracting sentiment scores. It assigns individual sentiment scores to each word in a sentence based on a sentiment dictionary. The sentiment dictionary is context-dependent. This means that for different contexts, the values for individual words in the dictionary change. For instance, the word "bug" in a movie review corpus is not treated as a negative term, whereas it is generally used negatively in a software engineering corpus. The dictionary in SentiStrength-SE is explicitly modified to incorporate software engineering terminology. It has been used as a state of the art tool for sentiment analysis studies in software engineering [101, 107].

Using SentiStrength-SE, each word is assigned a negative and positive score, ranging from -5 to -1 and +1 to +5 respectively. The whole sentence is assigned the minimum negative and maximum positive scores. The final score of the text is the sum of these two scores, ranging from -4 to +4. In this metrics, 0 is neutral emotion. The two poles demonstrate the severity of negative and positive emotions the smaller or larger the value is respectively. Table 6.1 shows sample scores of content from Mockito to depict the three emotions.

Table 6.1: Example of Sentiment Values in Issue Comments

No	Text	Score	Sentiment
1	No more ant!!!![+0.6 punctuation emphasis] Everybody rejoice[4]!!!!	+4	Positive
2	There was a lot of discussion about this feature in the corresponding PR: #1619	0	Neutral
3	Shadow plugin is a pain[-4] to use	-3	Negative

Sentiment analysis is built for natural text. Hence, before analyzing the sentiment, the text of the GitHub contents is cleaned. Contents can contain source code for pinpointing locations of bugs or other problems, markdown for formatting

text, as enabled by GitHub, html tags for further stylizing and urls for references. None of these are considered natural text, and so are detected and removed from the corpus.

6.2.3 Fix-Inducing Change Detection

To quantify the quality of code, Fix-Inducing Changes (FICs) are detected. FICs are code that introduce bugs to the system, inducing a later fix. FICs are detected from Commits in the following steps.

1. **Finding Fixes:** To detect FICs, first Commits that fix bugs are detected. To do so, Commit messages are analyzed to find terms like "bug", "fix" or "patch" [28], which depict bug fixing activities. If the changes in these Commits are (a) of source code and not documents or comments and (b) deleted or modified code, then these are labeled as Fixes.

2. **Tracking FIC:** For all the Fixes, each changed lines except insertions are listed. "Blame" is conducted on these lines to find the latest Commit to update these. Such Commits are listed as FICs, as these introduced code that Fixes removed or modified.

6.2.4 Dataset

The dataset comprises of fifty top Java projects from GitHub [113]. All **50 projects** are open source, enabling the extraction of Issues, Pulls and Commits, and all associated information. The projects contain, in total, **92,171 Issues** and **40,920 Pulls**. From these a total of approximately **500,000 comments** are fetched. Furthermore, the projects amass **217,205 Commits**.

6.3 Study Result

The following subsections contain description of how each of the two Sub-Research Questions (SRQs) are resolved. The descriptions include individual analysis processes and commentary on the findings.

6.3.1 SRQ1: Issue Sentiment

The first SRQ aims to observe the patterns of sentiment in GitHub Issues. This observation is conducted in several stages.

Firstly, it is observed how the sentiment scores — -4 to +4 — are distributed in Issues. All the texts in Issues are regarded individually to add to the frequency of associated sentiment scores. From a total of 495,892 text instances, which includes Issue title, body and comments, the resulting distribution is shown in Figure 6.3. It shows that more than half (58.4%) of the sentiment are neutral. In terms of polarity, positive sentiment outnumbers negative ones by 17%. Therefore, the average sentiment stands as **0.194**. This shows that, in the Issue-driven development process, most textual data are formal and task-centric, while the emotional messages tend to be more positive. This predominance of neutrality is in congruence with the result of the previous chapters where for both Commit messages and Pull contents, majority of the text contained non-emotional text.

Next the difference between Issues and Pulls is observed based on sentiment. Although the two contain similar structures — title, body and comments — their application goals are different, and hence are treated differently by contributors. For this observation, the texts are not individually tracked, rather, for each Issue or Pull, the average sentiment of all their text is extracted. From Figure 6.4, it is seen that Issues and Pulls have similar sentiment distribution. However, Pulls are more neutral, which is proven significant with a p-value of 0.0039, calculated from Wilcoxon rank sum test. This difference can be attributed to the fact that

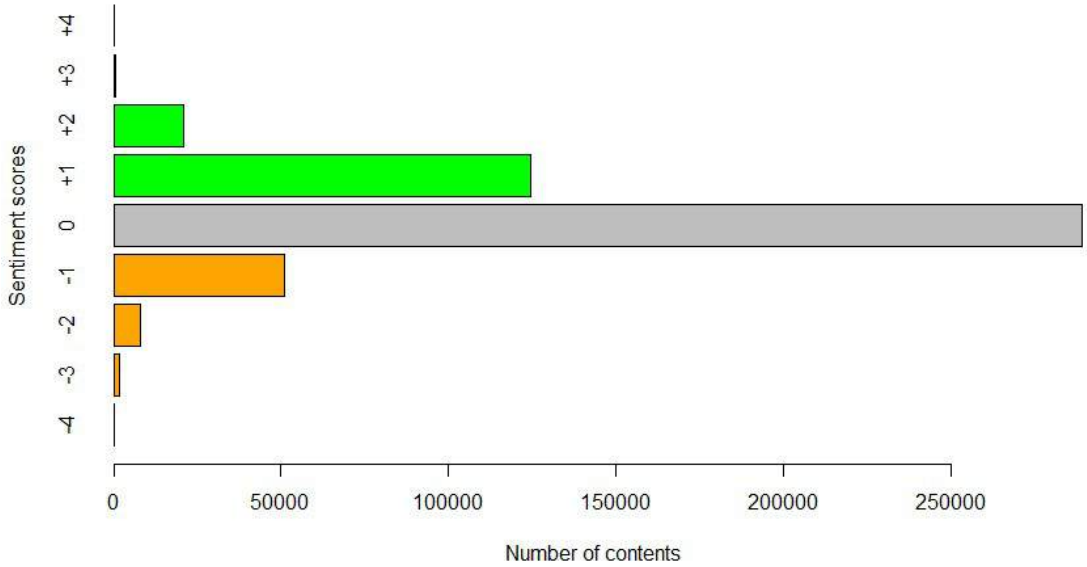

Figure 6.3: Sentiment score distribution in GitHub Issues

Issues contain less formal discussion than Pulls, with direct submission from end users and high level conversation about the problem among contributors. On the other hand, Pulls revolve around the developers' Commits and its modification, leading to source code-centric discussions.

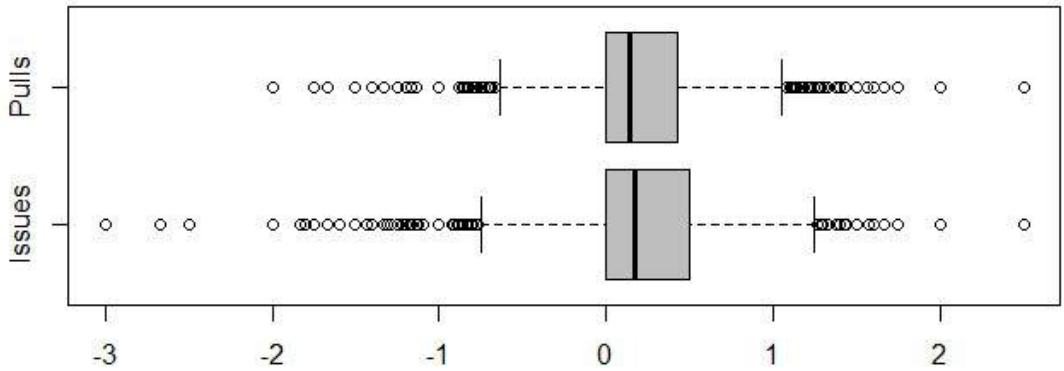

Figure 6.4: Box plots of sentiment in Issues and Pulls

6.3.2 SRQ2: Sentiment and FIC

SRQ2 analyzes the sentiment observed from the Issues and Pulls, and applies these to relate with Fix-Inducing Changes (FIC). After generating the artifact graph,

according to Section 6.2.1, Issues and their internal contents are directly linked to FICs through Pulls and Commits. It is observed that among the two types of links, 56% are through Pulls and 44% through Commits. Of the Issue-Pull links, 32% are rejected since those Pulls have not been merged, hence their Commits have been discarded.

To calculate sentiment, the two links are treated separately. For Issue-Pull links, sentiment is calculated for both the artifacts. For the other, only is Issue's average sentiment is extracted. Table 6.2 shows the proportions of sentiment from the resulting links, where -4 to -1 depicts Negative emotion and +1 to +4 Positive. It is seen that the artifacts are vastly dominated by positive sentiment, while neutrality is more prevalent than emotional text. The contents within the artifacts are analyzed separately as well, to understand their individual impact.

Table 6.2: Sentiment Proportions of Artifacts

	Polarity		**Subjectivity**	
Content	**Positive**	**Negative**	**Emotion**	**Neutral**
All	71%	29%	44%	56%
Title Only	54%	46%	24%	76%
Body Only	62%	38%	45%	55%
Title and Body	59%	41%	34%	66%
Comments Only	74%	26%	47%	53%

Using these combinations of artifacts, their relation with FICs are extracted. The artifacts are categorized based on their inclusion of Commits that are FICs.

Table 6.3 lists the findings after conducting Wilcoxon rank sum test on the resulting sentiment of categorized artifacts. It is seen that, when considering all the contents within artifacts, a significant relation exists between sentiment and FICs, with a p-value of $2.56E^{-06}$. It shows that sentiment of artifacts resulting in FICs are more positive than artifacts that do not introduce bugs.

Results of artifacts with filtered contents are also displayed in 6.3. For titles, there is no significant relation, although non-buggy artifacts are shown to be negative on average. The lack of relation is understandable when considering that

titles, from Table 6.2, are shown to have very little emotional text, with polar emotions being almost similarly distributed.

For the rest of the contents, a significant relation is observed. Issue or Pull bodies and comments tend to have more positive sentiment than negative. The combination of title and bodies also shows a significant relation to FICs, despite similar proportions to title-only sentiment. This combination is helpful to observe, as all Issues and Pulls, at their origin, contain these two contents. The subsequent discussion, along with the consequent code, can be highly influenced by these two original contents.

Table 6.3: SRQ2 Findings

	Average Sentiment		
Content	**FIC**	**Regular**	**p-value**
All	0.205	0.157	$2.56E^{-06}$
Title Only	0.008	-0.020	0.0923
Body Only	0.159	0.113	**0.0180**
Title and Body	0.084	0.047	**0.0082**
Comments Only	0.265	0.245	**0.0136**

*(bold values are $< \alpha$, where $\alpha = 0.05$)

6.3.3 SRQ3: Comparative Analysis

SRQ2 finalizes the correlation between developer sentiment and performance using the trinity of artifacts: Issues, Pulls and Commits. SRQ3 extends this finding to observe the varying effects of these artifacts and their contents.

First, the contents are grouped to contain different aspects of the artifacts. The grouping is listed as follows.

1. *pull-reviews*: In this group, only the Reviews from Pulls are considered. It was previously observed that positive and emotional Reviews are correlated to FICs.

2. *pull-comments*: This group considers the Comments from Pulls. Similar

to Reviews, positive and emotional Comments have been observed to be correlated to FICs.

3. *issue-comments*: Instead of Pulls, Comments from Issues are grouped here. The previous SRQ proved its correlation to FICs.

4. *issue-body*: Between the title and body of Issues, only bodies have shown to be significantly correlated to FICs. Hence, this grouping only considers the Issue bodies.

5. *commits*: Commits with emotional messages have shown to be more likely to be FICs. This group assembles all the Commit messages.

6. *all*: Similar to the previous SRQ, this group considers the Comments from Issues and Pulls along with their bodies to encapsulate contributor discussion under one term.

Next, statistical analysis is conducted to find the effect sizes of these groups. As the data is non-parametric — not normally distributed and with ordinal values — the popular method of Cohen's d [114], which requires the data to be parametric, cannot be applied. Instead Wilcoxon r [115] is used according to equation 6.1.

$$r = \frac{Z}{\sqrt{N}}, \tag{6.1}$$

where Z is the statistical value from the Wilcoxon test and N is the total number of observations.

The observations, as presented in Figure 6.5, are as follows.

1. **Pulls**: Both the contents from Pulls — Comments and Reviews — display comparatively large effect sizes. Moreover, Comments shows the highest among the six groups.

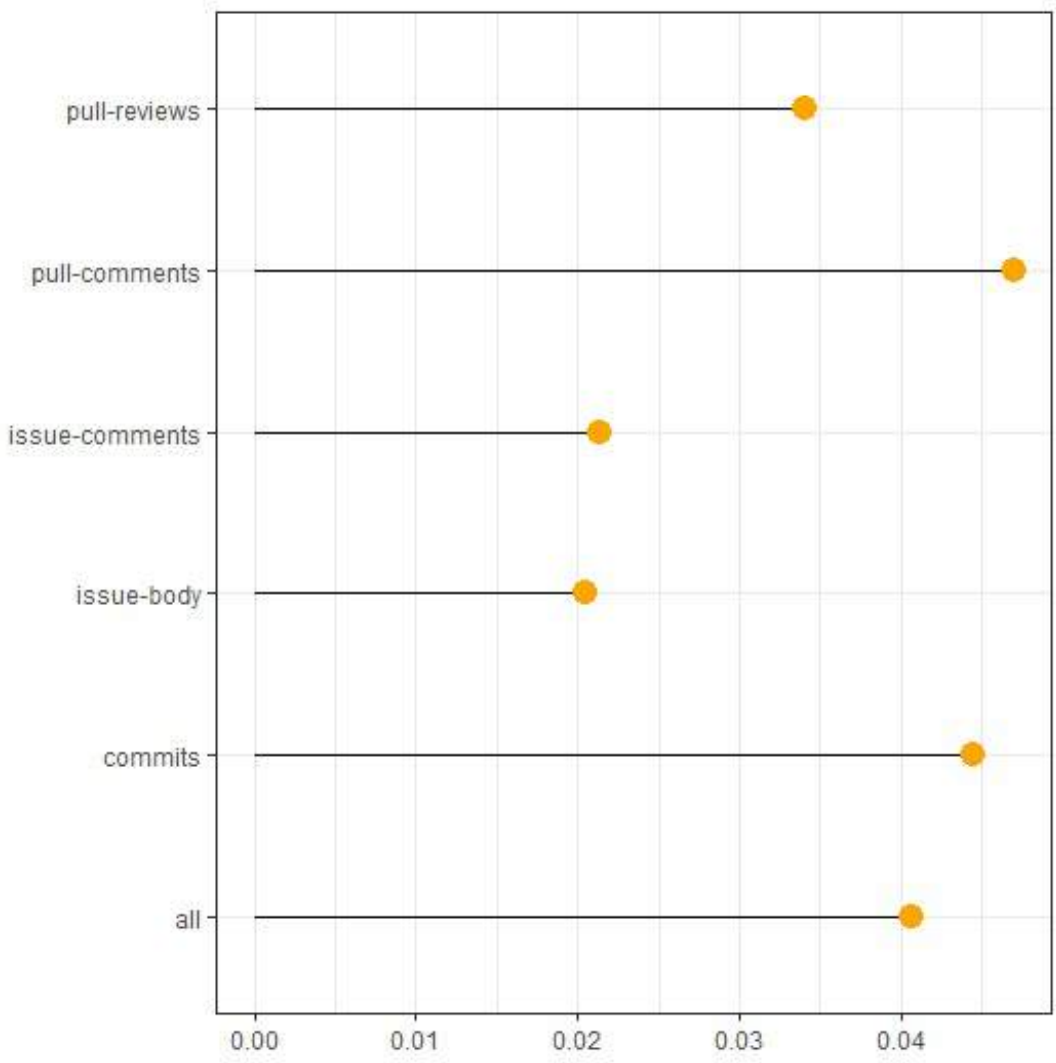

Figure 6.5: Comparison of effect sizes from the different groups of contents

2. **Issues**: The contents of Issues show the lowest effect sizes. Both Comments and bodies range near 0.02 in terms of r values.

3. **Commits**: Commit messages are observed to also contain a comparatively large effect size. Although the messages are seen to be mostly neutral, the minimal differences prove to create the effect, making it the second highest of the groups.

4. **Combined**: From the values, it is seen that the combined contents perform highly as well. It is not the highest, which can be contributed to its inclu-

sion of Issue Comments and bodies, but it ranks third among the groups. Therefore, contributor discussions as a whole, in lieu of separate considerations of different contents, can be applied to understand the relation between developer sentiment and their performance.

The findings from the three SRQs can be applied in the following ways:

- Whether bugs are likely to be introduced from an Issue can be estimated from the sentiment of related contents. Hence, in the presence of positive or more emotional discussion in Issues and Pulls, review efforts can be emphasized for the resulting Commits.

- Since emotional discussion outnumbers neutral ones in bug-inducing Issues, the tone of conversation can be scrutinized from a project management point of view.

6.4 Threats to Validity

While the study conducts its analysis on a vast amount of data, with the total size of contents — Issues, Pulls, comments and Commits — being approximately 1,000,000, the projects are all written in Java. Whether the results can vary based on project language has not been analyzed.

The sentiment of the contents is dependent on the score calculated by SentiStrength-SE, which has been trained on Jira Issue comments and do not have a 100% accuracy. However, no sentiment analysis tool has yet produced such results due to the ambiguity of natural text.

Questions can be raised on the viability of the FIC detection process. Although not manually evaluated in this study, the literature has established this as the standard mechanism to detect the introduction of bugs.

6.5 Summary

This study analyzed the sentiment in GitHub Issues to relate contributor sentiment to the quality of code. Fix-Inducing Changes (FICs) have been used as a metric to determine the quality of code as these indicate source code changes that introduce bugs. By creating an artifact relation graph between Issues, Pulls and Commits, direct links have been generated between contributor discussions and the resulting code. It is observed that contributors write mostly neutral text in the artifacts, while their emotional messages tend to be more positive than negative. A relation is observed between the average sentiment of the discussions and FICs, where positive discussion is shown to lead to buggy code. Lastly, it is seen that different contents have different effect on the correlation, with the combined contents representing contributor discussion having an equally strong effect size.

Chapter 7

Conclusion

The application of sentiment analysis in the domain of software engineering has seen a surge in recent research efforts due to its unique and insightful new perspective. This perspective utilizes the social aspect enabled in modern software development processes [22]. Using collaborative artifacts in online software project management systems, contributors converse among each other and contribute to the software [3]. Textual information provided by the collaborative artifacts are used for extracting sentiment of the contributors. Previous studies adopted this process to observe patterns and correlations related to contributor sentiment, that include: characteristics of the project [5], properties of development tasks [5, 6], team distribution [5, 10], developer productivity [7, 8, 9, 10, 4], external factors [9, 27] and task outcome [31]. However, to the best of the knowledge, whether the developers' performance is affected by sentiment has not been studied yet. To alleviate this, this research conducts a study to correlate contributor sentiment to developer performance. As a metric of performance, Fix-Inducing Changes (FICs) have been used, which represent changes to code that introduce bugs to the system [11]. Table 7.1 lists the individual properties observed in literature compared to the one proposed in this thesis. This correlation has been conducted in three separate parts. First, it is observed whether developer contribution itself provides

impactful sentiment. Then, the development task, consisting of contribution and reviews, is analyzed. Lastly, conversation among contributors are explored. This chapter describes in brief the contribution and achievements of these three studies, and concludes with a road-map for future work in this domain.

Table 7.1: Comparison of experimentations conducted in literature and proposed in this thesis

Paper	**Properties Observed**	**Metrics Used**	**Artifacts Used**
Guzman et al. [27]	Task characteristics	Deadlines	E-mails, commits, bug reports and wikis
Garcia et al. [9]	Productivity	Developer inactivity	Bugzilla reports
	External factors	Core contributor turnover	Bugzilla reports
Guzman et al. [5]	Project and task characteristics	Project language, day and time of submission	GitHub Commits
	Team distribution	geographic distribution of developers	
Sinha et al. [6]	Task characteristics	Day and time of submission	GitHub Commits
	Task size	Number of files changed in single Commit	
Souza et al. [31]	Task outcome	Build success/failure	GitHub Commits
Jurado et al. [4]	Productivity	Issue activities	GitHub Issues
Ortu et al. [7]	Productivity	Issue resolution time	Jira Issues
Mäntlyä et al. [8]	Productivity	Issue resolution time	Jira Issues
El Asri et al. [10]	Productivity	Review resolution time	Gerrit Reviews
	Team distribution	Reviewer position	Gerrit Reviews
Proposed in this thesis	Performance	Fix-Inducing Changes	Commits, Pull Requests and Issues

7.1 Sentiment in Developer Contribution and FIC

In the first step, a correlation between the sentiment in developer contribution and FICs is established. Developer contribution is defined as a contributor's modification of code. To analyze contribution, Commits [24] from GitHub are used. In Commits, the author adds a message that describes the changes made. This message is used as a textual input for sentiment analysis.

For correlating the sentiment with bug-related changes, four types of Commits are categorized:

1. Fix-Inducing Changes (FIC): Commits that create bugs,
2. parent of FICs (pFIC): Commits that directly precede a buggy change,
3. Fixing Changes (FC): Commits that fix bugs, and
4. Fix-inducing Fixes (FIF): Commits that fix bugs but create new ones.

Sentiment polarity values of the messages are extracted using the classifier Senti4SD [85]. Polarity values include *positive*, *negative* and *neutral* emotions. The dataset for this analysis is constructed using thirteen Java repositories from GitHub [102]. These amassed a total of 171,971 Commits for analysis. From the fetched dataset, each category of Commit is statistically differentiated with regular Commits using Wilcoxon rank-sum test and Chi-square test of independence.

From the statistical analysis, it is seen that messages in bug-related Commits are more negative than regular Commits. For instance, the average sentiment for messages in FCs is -0.119 compared to -0.058 for regular ones, with a p-value less than $8.8e^{-16}$ depicting statistical significance. Bug-related Commits also contain more instances of negative and emotional messages. For example, FICs — buggy code — contain 6% more emotional messages than regular Commits. Lastly, a differentiation between correct and incorrect fixes is derived based on sentiment. FIFs are more negative (0.31 on average) and contain more emotional texts (5%

more), than FCs that are not FIFs. These differences, as well, are calculated as statistically significant.

This result differentiates Commits that are related to bugs from regular ones based on developer sentiment. The finding strengthens the patterns observed in previous studies and prompts further investigation into the quantifiable impact of emotions on developer performance.

7.2 Sentiment in Development Tasks and FIC

Next, the analysis extends to the context of development tasks, which include commentary and reviews on submitted code. Pull Requests [26] in GitHub encompass these activities in the form of three artifacts:

1. Commits: the contribution
2. Comments: high-level commentary on task resolution
3. Reviews: code-specific feedback of the Commits

These artifacts are fetched using the GitHub API [92]. Six popular java projects [109] are used for extracting the data. The dataset comprised of a total of 6619 Pull Requests. From the 78,220 Commits, FICs are extracted to represent buggy code.

Sentiment from these artifacts are calculated using SentiStrength-SE [84], which provides a sentence or text-level sentiment polarity score ranging from -4 to +4, 0 being the neutral value. A Commit in a Pull Request is set a sentiment value which is the sum of sentiment scores of other Commits, Comments and Reviews that precede it. This represents the cumulative emotional state which led up to the Commit. Two sets — one of regular Commits and the other FICs — are generated using this process. These two sets are statistically differentiated using Wilcoxon rank sum test.

The findings show that there are significant differences of sentiment scores between FICs and regular Commits. Comments and Reviews before FICs have a more positive sentiment than those before regular ones. Compared to 86.80% positive Comments and 59.5% positive Reviews before FICs, regular Commits contain 63.9% and 31.2% positive ones respectively. Conversely, Commits and the overall discussions preceding FICs are more negative than ones preceding regular Commits. For all three artifacts, neutrality is significantly lower before FICs. Commits, Comments and Reviews contain 21.4%, 16.8% and 23.7% less neutral text respectively before FICs compared to regular Commits. This information suggests that whether bugs are being introduced in the system can be indicated through the emotions of collaborative artifacts that precede a change.

7.3 Sentiment in Contributor Discussion and FIC

The third and last part of this research extends sentiment analysis to the contributor discussions that occur before the start of development tasks. These discussions are found in project Issues [25], which are reports from end users and contributors describing bugs, feature requests and more. These reports enable discussion among contributors regarding the issue at hand. Their discussion is used as a guideline in the resulting development task, which change the software with its Commits. Hence, sentiment in the Issue discussion is analyzed to observe its impact on the quality of the resulting code.

The dataset for this study is constructed using 50 Java projects from GitHub [113]. The resulting dataset contains 92,171 Issues, 40,920 Pull Requests and approximately 500,000 Comments. Because of the lack of direct connections, a graph is generated that connects the Issues with Pull Requests and Commits utilizing Issue reference IDs [76].

From the resulting graph, Issues and its Comments can now be linked to project

Commits. From the 217,205 Commits fetched from the dataset, FICs are detected. These are categorized as buggy Commits and the rest regular ones.

The sentiment of the Comments is calculated using SentiStrength-SE [84]. From the sentiment of a total of 495,892 text instances, which includes Issue title, body and comments, neutral sentiment is the prevailing one with a ratio of 58.4%. In terms of polarity, instances of positive text are 17% more than negative ones, with an average sentiment of 0.194. Compared to Pull Requests, Issues are less neutral with a significant p-value of 0.0039, indicating a lessened formality when discussing about problems from a high level perspective.

In the correlation analysis, Wilcoxon rank sum test is used from checking statistical significance. It is observed that a statistically significant difference exists in the sentiment between Issues that lead up to bugs and Issues that do not, with a p-value of $2.56e^{-06}$. Issue Comments contain the most positive text instances (74%) out of the other contents while titles contain the most neutral ones (76%). The analysis shows that sentiment of artifacts resulting in FICs are more positive than artifacts that do not introduce bugs. This is similar to the results from sentiment in development tasks.

Lastly, in the effect size comparison, six different groups are categorized using contents from the three major artifacts: Issues, Pulls and Commits. Wilcoxon r is applied to measure the effect sizes. It is seen that separately the contents have different effects, with contents from Pulls and Commits performing better than those of Issues. On the other hand, when the contents are combined, the effect size is similar to the higher ranked groups.

These results indicate that sentiment from collaborative artifacts can be used to estimate developer performance. It can also be used as a guideline to monitor the language used in development-related discussions.

7.4 Future Work

This research is a primary exploration of the correlation between sentiment in collaborative artifacts and developer performance. From the processes and findings of this research, extensions can be studied in the following ways:

1. **Utilizing artifacts outside of the scope of development processes**: This research encapsulates the software development process by analyzing development artifacts like Commits, Issues and Pull Requests. Sentiment analysis, however, can be applied beyond the scope of development processes. A popular source of textual data for analyzing sentiment has been social media. Future studies, informed by the findings of this research, can incorporate artifacts fetched from social media, outside of the purview of development activities. Guzman et al. [27] mentioned the applicability of Twitter and other social media posts from the developers as a way to understand their unfiltered mood during development processes. In alignment to this research, future work can therefore integrate expressions of developers from public contents like social media posts, email in mailing list and more, and relate these to their tasks at the time.

2. **Empirically evaluating sentiment**: The scope of this study has been to incorporate repository mining with sentiment analysis to for analyzing developer performance. This enabled a time-efficient exploration of a large corpus of contents that would not be possible via empirical studies. However, empirical studies, albeit in much smaller scope, can evaluate the findings with real world circumstances, as demonstrated by Guzman et al. [27]. Furthermore, by assessing the sentiment detected with the author of the text, the correctness of analysis can be better evaluated. The absence of a gold standard has been mentioned as a shortcoming in this field of work [9, 8, 10]. Empirical evaluation can alleviate that to an extent. In this line

of research, sentiment extracted from project development artifacts can be compared with the direct responses — through interviews or surveys — with participants of that project. And based on the results of this research, these surveys can be used to track or indicate buggy activities.

3. **Exploring social dynamics**: In the workplace, social structures have been observed to play a significant role in differentiating interactions, with employees expressing adjusted sentiment based on job roles [20]. Social dynamics is also a key aspect in modern software development environments [22]. El Asri et al. [10] observed that sentiment from reviewers differ based on their roles — core vs peripheral. Furthermore, negativity from peripheral reviewers affected review acceptance time more than negativity from core reviewers. While this study considered developers as single independent units, the results that it produced — an association between developer sentiment and buggy code — can be analyzed in fields of development collaboration by considering social and structural hierarchies. The difference in performance and sentiment expression based on individual roles and interaction between developers of different positions can be investigated.

4. **Automating real-time correlations via bots**: The studies in this field ultimately assist in software development management by providing insight into the effects of developer sentiment. GitHub bots have been developed that display emotions of developers in collaborative artifacts [116]. These bots can be integrated with the correlations observed in this field, for instance, properties of the project and task [5, 6], developer productivity [7, 8, 9, 10], team distribution [5, 10], and more, including the findings of this study. The bot can extract emotions from posted text from contributors and display whether buggy code is likely to be written. For instance, since it is seen that non-neutral Comments and Reviews in Pull Requests can lead

to bug, a bot in Pull Requests can calculate the collective sentiment of the artifacts and notify a maintainer that Commits in this Pull Request may contain bugs. This would directly impact software development processes with the benefits of this study.

Bibliography

[1] B. Pang, L. Lee, *et al.*, "Opinion mining and sentiment analysis," *Foundations and Trends® in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[2] K. Herzig and A. Zeller, *Mining Your Own Evidence*, ch. 27. O'Reilly Media, Inc., Oct. 2010.

[3] D. Berry, *The Philosophy of Software: Code and Mediation in the Digital Age*. Palgrave, 01 2011.

[4] F. Jurado and P. Rodriguez, "Sentiment analysis in monitoring software development processes: An exploratory case study on github's project issues," *Journal of Systems and Software*, vol. 104, pp. 82–89, 2015.

[5] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 352–355, ACM, 2014.

[6] V. Sinha, A. Lazar, and B. Sharif, "Analyzing developer sentiment in commit logs," in *Proceedings of the 13th International Conference on Mining Software Repositories*, pp. 520–523, ACM, 2016.

[7] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive?: empirical study of affectiveness vs. issue fixing time," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, pp. 303–313, IEEE Press, 2015.

[8] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity?," in *Proceedings of the 13th International Conference on Mining Software Repositories*, pp. 247–258, ACM, 2016.

[9] D. Garcia, M. S. Zanetti, and F. Schweitzer, "The role of emotions in contributors activity: A case study on the gentoo community," in *2013 International Conference on Cloud and Green Computing*, pp. 410–417, IEEE, 2013.

[10] I. El Asri, N. Kerzazi, G. Uddin, F. Khomh, and M. J. Idrissi, "An empirical study of sentiments in code reviews," *Information and Software Technology*, 2019.

[11] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," in *ACM sigsoft software engineering notes*, vol. 30, pp. 1–5, ACM, 2005.

[12] M. Monperrus, "Automatic software repair: a bibliography," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–24, 2018.

[13] S. Kim and E. J. Whitehead Jr, "How long did it take to fix bugs?," in *Proceedings of the 2006 international workshop on Mining software repositories*, pp. 173–174, 2006.

[14] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," in *2009 IEEE International Conference on Software Maintenance*, pp. 523–526, IEEE, 2009.

[15] L. Yue, W. Chen, X. Li, W. Zuo, and M. Yin, "A survey of sentiment analysis in social media," *Knowledge and Information Systems*, pp. 1–47, 2019.

[16] N. Godbole, M. Srinivasaiah, and S. Skiena, "Large-scale sentiment analysis for news and blogs.," *Icwsm*, vol. 7, no. 21, pp. 219–222, 2007.

[17] P. Melville, W. Gryc, and R. D. Lawrence, "Sentiment analysis of blogs by combining lexical knowledge with text classification," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1275–1284, 2009.

[18] H. Tang, S. Tan, and X. Cheng, "A survey on sentiment detection of reviews," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10760–10773, 2009.

[19] R. Gelbard, R. Ramon-Gonen, A. Carmeli, R. M. Bittmann, and R. Talyansky, "Sentiment analysis in organizational work: Towards an ontology of people analytics," *Expert Systems*, vol. 35, no. 5, p. e12289, 2018.

[20] M. De Choudhury and S. Counts, "Understanding affect in the workplace via social media," in *Proceedings of the 2013 conference on Computer supported cooperative work*, pp. 303–316, ACM, 2013.

[21] "Compare repositories - open hub." https://www.openhub.net/repositories/compare. Accessed: 2020-08-30.

[22] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pp. 1277–1286, 2012.

[23] "The state of the octoverse." https://octoverse.github.com/. Accessed: 2020-08-30.

[24] "About commits." https://docs.github.com/en/github/committing-changes-to-your-project/about-commits. Accessed: 2020-08-30.

[25] "About issues." https://docs.github.com/en/github/managing-your-work-on-github/about-issues. Accessed: 2020-08-30.

[26] "About pull requests." https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests. Accessed: 2020-08-30.

[27] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pp. 671–674, ACM, 2013.

[28] S. Kim, E. J. Whitehead Jr, and Y. Zhang, "Classifying software changes: Clean or buggy?," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.

[29] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram, "How do fixes become bugs?," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 26–36, ACM, 2011.

[30] A. Z. Sadiq, M. J. I. Mostafa, and K. Sakib, "On the evolutionary relationship between change coupling and fix-inducing changes," in *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 494–501, SCITEPRESS-Science and Technology Publications, Lda, 2019.

[31] R. Souza and B. Silva, "Sentiment analysis of travis ci builds," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 459–462, IEEE, 2017.

[32] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th working conference on mining software repositories*, pp. 262–271, 2014.

[33] "Build software better, together." https://github.com/. Accessed: 2020-08-30.

[34] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[35] M. Tsytsarau and T. Palpanas, "Survey on mining subjective data on the web," *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 478–514, 2012.

[36] B. Liu and L. Zhang, "A survey of opinion mining and sentiment analysis," in *Mining text data*, pp. 415–463, Springer, 2012.

[37] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. J. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the workshop on language in social media (LSM 2011)*, pp. 30–38, 2011.

[38] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment," in *Fourth international AAAI conference on weblogs and social media*, Citeseer, 2010.

[39] A. Bhatt, A. Patel, H. Chheda, and K. Gawande, "Amazon review classification and sentiment analysis," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 6, pp. 5107–5110, 2015.

[40] J. Hornikx and B. Hendriks, "Consumer tweets about brands: A content analysis of sentiment tweets about goods and services," *Journal of Creative Communications*, vol. 10, no. 2, pp. 176–185, 2015.

[41] T. T. Thet, J.-C. Na, and C. S. Khoo, "Aspect-based sentiment analysis of movie reviews on discussion boards," *Journal of information science*, vol. 36, no. 6, pp. 823–848, 2010.

[42] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd international requirements engineering conference (RE)*, pp. 153–162, IEEE, 2014.

[43] R. Plutchik and H. Kellerman, *Emotion, theory, research, and experience.* Academic press, 1980.

[44] W. G. Parrott, *Emotions in social psychology: Essential readings.* Psychology Press, 2001.

[45] D. Graziotin, X. Wang, and P. Abrahamsson, "Understanding the affect of developers: theoretical background and guidelines for psychoempirical software engineering," in *Proceedings of the 7th International Workshop on Social Software Engineering*, pp. 25–32, 2015.

[46] R. D. Lane, P. M. Chua, and R. J. Dolan, "Common effects of emotional valence, arousal and attention on neural activation during visual processing of pictures," *Neuropsychologia*, vol. 37, no. 9, pp. 989–997, 1999.

[47] P. J. Lang, M. K. Greenwald, M. M. Bradley, and A. O. Hamm, "Looking at pictures: Affective, facial, visceral, and behavioral reactions," *Psychophysiology*, vol. 30, no. 3, pp. 261–273, 1993.

[48] K. Lewin, "A dynamic theory of personality: Selected papers," *The Journal of Nervous and Mental Disease*, vol. 84, no. 5, pp. 612–613, 1936.

[49] M. M. Bradley and P. J. Lang, "Measuring emotion: the self-assessment manikin and the semantic differential," *Journal of behavior therapy and experimental psychiatry*, vol. 25, no. 1, pp. 49–59, 1994.

[50] C. Lutz and G. M. White, "The anthropology of emotions," *Annual review of anthropology*, vol. 15, no. 1, pp. 405–436, 1986.

[51] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[52] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 168–177, 2004.

[53] S.-M. Kim and E. Hovy, "Determining the sentiment of opinions," in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pp. 1367–1373, 2004.

[54] A. Fahrni and M. Klenner, "Old wine or warm beer: Target-specific sentiment analysis of adjectives," *Affective Language in Human and Machine AISB*, 2008.

[55] J. Read and J. Carroll, "Weakly supervised techniques for domain-independent sentiment classification," in *Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, pp. 45–52, 2009.

[56] I. Maks and P. Vossen, "A lexicon model for deep sentiment analysis and opinion mining applications," *Decision Support Systems*, vol. 53, no. 4, pp. 680–688, 2012.

[57] W. Zhang, H. Xu, and W. Wan, "Weakness finder: Find product weakness from chinese reviews by using aspects based sentiment analysis," *Expert Systems with Applications*, vol. 39, no. 11, pp. 10283–10291, 2012.

[58] H. Kang, S. J. Yoo, and D. Han, "Senti-lexicon and improved naïve bayes algorithms for sentiment analysis of restaurant reviews," *Expert Systems with Applications*, vol. 39, no. 5, pp. 6000–6010, 2012.

[59] J. Ortigosa-Hernández, J. D. Rodríguez, L. Alzate, M. Lucania, I. Inza, and J. A. Lozano, "Approaching sentiment analysis by using semi-supervised learning of multi-dimensional classifiers," *Neurocomputing*, vol. 92, pp. 98–115, 2012.

[60] C. C. Chen and Y.-D. Tseng, "Quality evaluation of product reviews using an information quality framework," *Decision Support Systems*, vol. 50, no. 4, pp. 755–768, 2011.

[61] R. Moraes, J. F. Valiati, and W. P. G. Neto, "Document-level sentiment classification: An empirical comparison between svm and ann," *Expert Systems with Applications*, vol. 40, no. 2, pp. 621–633, 2013.

[62] A. Aue and M. Gamon, "Customizing sentiment classifiers to new domains: A case study," in *Proceedings of recent advances in natural language processing (RANLP)*, vol. 1, pp. 2–1, Citeseer, 2005.

[63] T. Pearse and P. Oman, "Maintainability measurements on industrial source code maintenance activities," in *Proceedings of International Conference on Software Maintenance*, pp. 295–303, IEEE, 1995.

[64] D. Spinellis, "Version control systems," *IEEE Software*, vol. 22, no. 5, pp. 108–109, 2005.

[65] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[66] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, *Version control with subversion: next generation open source version control.* " O'Reilly Media, Inc.", 2008.

[67] P. Cederqvist, R. Pesch, *et al.*, "Version management with cvs," 1992.

[68] B. O'Sullivan, *Mercurial: The Definitive Guide: The Definitive Guide.* " O'Reilly Media, Inc.", 2009.

[69] M. Biazzini, M. Monperrus, and B. Baudry, "On analyzing the topology of commit histories in decentralized version control systems," in *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 261–270, IEEE, 2014.

[70] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pp. 23–32, IEEE, 2003.

[71] "Jira software - features — atlassian." https://www.atlassian.com/software/jira/features. Accessed: 2020-08-30.

[72] "Features :: Bugzilla :: bugzilla.org." https://www.bugzilla.org/features/. Accessed: 2020-08-30.

[73] "The first single application for the entire devops lifecycle - gitlab — gitlab." https://about.gitlab.com/. Accessed: 2020-08-30.

[74] "Bitbucket — the git solution ofor professional teams." https://bitbucket.org/. Accessed: 2020-08-30.

[75] "mockito/mockito: Most popular mocking framework for java." https://github.com/mockito/mockito. Accessed: 2020-08-30.

[76] "Autolinked references and urls." https://help.github.com/github/writing-on-github/autolinked-references-and-urls. Accessed: 2020-08-30.

[77] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24003–24015, 2018.

[78] C. Padmini, "Beginners guide to software testing," *Indianapolis: Lulu Enterprises, Inc*, 2006.

[79] I. ISO, "Ieee, systems and software engineering–vocabulary," *IEEE computer society, Piscataway, NJ*, vol. 8, p. 9, 2010.

[80] S. Desikan and G. Ramesh, *Software testing: principles and practice*. Pearson Education India, 2006.

[81] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[82] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the American society for information science and technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[83] C. Strapparava, A. Valitutti, *et al.*, "Wordnet affect: an affective extension of wordnet.," in *Lrec*, vol. 4, p. 40, Citeseer, 2004.

[84] M. R. Islam and M. F. Zibran, "Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text," *Journal of Systems and Software*, vol. 145, pp. 125–146, 2018.

[85] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018.

[86] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: a customized sentiment analysis tool for code review interactions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 106–111, IEEE, 2017.

[87] "Confluence - accomplice more together — atlassian." http://www.atlassian.com/software/confluence. Accessed: 2020-08-30.

[88] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[89] "Gentoo's bugzilla main page." https://bugs.gentoo.org/. Accessed: 2020-08-30.

[90] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[91] M. Beller, G. Gousios, and A. Zaidman, "Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 447–450, IEEE, 2017.

[92] "Github api v3 — github developer guide." https://developer.github.com/v3/. Accessed: 2020-08-30.

[93] M. M. Bradley and P. J. Lang, "Affective norms for english words (anew): Instruction manual and affective ratings," tech. rep., Technical report C-1, the center for research in psychophysiology . . . , 1999.

[94] T. Wilson, P. Hoffmann, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan, "Opinionfinder: A system for subjectivity analysis," in *Proceedings of HLT/EMNLP 2005 Interactive Demonstrations*, pp. 34–35, 2005.

[95] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pp. 52–56, 2010.

[96] A. B. Warriner, V. Kuperman, and M. Brysbaert, "Norms of valence, arousal, and dominance for 13,915 english lemmas," *Behavior research methods*, vol. 45, no. 4, pp. 1191–1207, 2013.

[97] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," in *Proceedings of the 13th International Conference on Mining Software Repositories*, pp. 460–463, 2016.

[98] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? an empirical study," in *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, pp. 104–113, IEEE, 2012.

[99] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 550–554, IEEE, 2015.

[100] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–8, 2011.

[101] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pp. 364–375, IEEE, 2018.

[102] "Top 11 popular java projects on github." https://medium.com/issuehunt/top-11-popular-java-projects-on-github-48aaad5b4e0a. Accessed: 2020-08-30.

[103] "Msr 2014 mining challenge dataset." http://ghtorrent.org/msr14.html. Accessed: 2020-08-30.

[104] R. J. Jones, S. A. Woods, and Y. R. Guillaume, "The effectiveness of workplace coaching: A meta-analysis of learning and performance outcomes from coaching," *Journal of Occupational and Organizational Psychology*, vol. 89, no. 2, pp. 249–277, 2016.

[105] "jcabi-github - objectoriented github java api." https://github.jcabi.com/. Accessed: 2020-08-30.

[106] "Jgit." https://github.com/eclipse/jgit. Accessed: 2020-08-30.

[107] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 94–104, IEEE, 2018.

[108] "Diffj." https://github.com/jpace/diffj. Accessed: 2020-08-30.

[109] "Top 11 popular java projects on github." https://medium.com/issuehunt/top-11-popular-java-projects-on-github-48aaad5b4e0a. Accessed: 2020-08-30.

[110] "Laboratory for software engineering research." https://laser.cs.uno.edu/Projects/Projects.html. Accessed: 2020-08-30.

[111] K. Peterson, "The github open source development process," *url: http://kevinp. me/github-process-research/github-processresearch. pdf (visited on 05/11/2017)*, 2013.

[112] "Fork a repo." https://docs.github.com/en/github/getting-started-with-github/fork-a-repo. Accessed: 2020-08-30.

[113] "50 top java projects on github." https://medium.com/issuehunt/50-top-java-projects-on-github-adbfe9f67dbc. Accessed: 2020-08-30.

[114] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[115] C. O. Fritz, P. E. Morris, and J. J. Richler, "Effect size estimates: current use, calculations, and interpretation.," *Journal of experimental psychology: General*, vol. 141, no. 1, p. 2, 2012.

[116] "Github sentiment bot." https://github.com/apps/sentiment-bot. Accessed: 2020-08-30.