"@alex, this fixes #9": Analysis of Referencing Patterns in Pull Request Discussions

by

Ashish Chopra

B.Tech. I.T., Guru Gobind Singh Indraprastha University, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

February 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

"@alex, this fixes #9": Analysis of Referencing Patterns in Pull Request Discussions

submitted by **Ashish Chopra** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Dongwook Yoon, Computer Science
Supervisor

Ivan Beschastnikh, Computer Science
Supervisory Committee Member

Sidney S. Fels, Electrical and Computer Engineering
Supervisory Committee Member

Abstract

Pull Requests (PRs) are a frequently used method for proposing changes to source code repositories. When discussing proposed changes in a PR discussion, stakeholders often reference a wide variety of information objects for establishing shared awareness and common ground. Previous work has not considered how the referential behavior impacts collaborative software development via PRs. This knowledge gap is the major barrier in evaluating the current support for referencing in PRs and improving them. We conducted an explorative analysis of $\sim$7K references, collected from 450 public PRs on GitHub, and constructed taxonomies of referent types and expressions. Using our annotated dataset, we identified several patterns in the use of references. We found that despite a prevalent use of references in PR discussions, GitHub's interface lacks the support for referencing the majority of information types. We provide qualitative descriptions of how different contextual factors shape the use of references in discussions. We also discovered distinct referencing patterns in merged and closed PRs which signifies a potential ground for future research to establish a relationship between reference use and PR outcomes. These findings suggest that what is and is not referenced within a PR discussion has an important impact on the software development process, and warrants continued platform support and research. We conclude with design implications to support more effective referencing in PR discussion interfaces.

Lay Summary

In software development, pull request (PR) is a process of submitting and reviewing code changes to software repositories. During the PR review, people often reference a variety of information resources such as code-related elements (e.g., variables, functions, classes) and non-code artifacts (e.g., users, docs, web URLs) in their text comments. In this research, we analyzed a dataset of 7k references collected from 450 PR threads on GitHub to understand what information resources people mention and how they are expressed. We also identified several referencing patterns used in discussions. We found that the use of references is prevalent in PR discussions, but GitHub interface lacks the support for referencing a majority of information types. We described how contextual factors of PR discussion shape people's referencing practices. We also discovered distinct referencing patterns in merged and closed PRs which signifies a potential ground for further research. Based on our findings, we proposed implications for designing effective referencing support in PR discussion interfaces.

Preface

This thesis is an original intellectual product of the author, A. Chopra. All of the work presented henceforth was conducted in D-Lab, located in the department of Computer Science in UBC, headed by Prof. Dongwook Yoon.

The research methods described in Chapter 3 was designed and conducted in close collaboration with Prof. Dongwook Yoon, Prof. Ivan Bestchastnikh, Prof. Sidney S. Fels, Samuel Dodson (PhD candidate), and Morgan Mo (undergraduate research intern).

Ashish Chopra and Morgan Mo conducted the qualitative coding of PRs. Morgan Mo designed and implemented the data extraction utility used for compiling the references dataset as described in Section 3.3.1.

The Correspondence Analysis and Hierarchical Clustering described in Section 3.3.2 was designed and implemented by Samuel Dodson.

Table of Contents

List of Tables

List of Figures

List of Supplementary Materials

PR_references_dataset.csv A compiled dataset of ∼7k references in CSV format.

Glossary

API	Application Programming Interface
CSCW	Computer-Supported Cooperative Work
CSV	Comma-separated Values
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HCI	Human-Computer Interaction
JSON	JavaScript Object Notation
PR	Pull Request
QCA	Qualitative Content Analysis
UI	User Interface
URL	Uniform Resource Locator

Acknowledgments

I wish to acknowledge my supervisor, Prof. Dongwook Yoon, and the members of supervisory committee Prof. Ivan Beschastnikh and Prof. Sidney S. Fels for their constant support and guidance from the time of inception to the conclusion of this research work which enormously helped in shaping it to what it is today.

Additionally, I would like to acknowledge Samuel Dodson for providing valuable review on a frequent basis, and Morgan Mo for contributing hours of coding qualitative data and developing the data extraction pipeline. I would also like to recognize the following people for their feedback and review during various stages of the project: Anna Offenwanger, Mint Tanprasert, Dr. Ning Ma, Prof. Kyoungwon Seo, Joice Tang, Mohi Reza, Frances Sin, Matthew Fong, Paul Bucci, Prof. Ido Roll.

I would also like to acknowledge the financial support provided by Huawei and the NSERC CREATE program, Designing for People research cluster of UBC which made this research work possible.

Finally, I would like to express my gratitude to my parents and my beloved sister for their constant motivational support during this journey.

Chapter 1

Introduction

Asatoma Sadgamaya — Brhadaranyaka Upanishad (1.3.28)

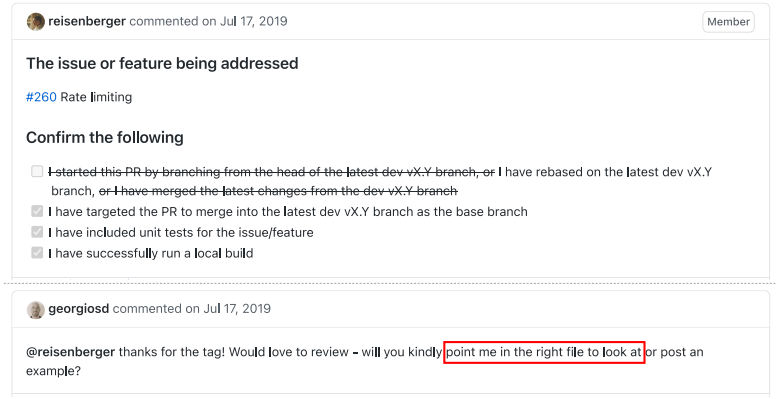

Figure 1.1: This figure shows the slice of discussion from a PR to showcase common ground failure due to absence of a reference highlighted in red box. The grey dashed line indicates abridged content not shown in the capture for brevity.

Making a pull request (PR) [1] is the primary method for proposing changes to source code repositories [26]. GitHub reported that more than 87 million PRs were merged on its platform between 2018 and 2019 [2]. Discussion among stakeholders is often a crucial aspect of reviewing a proposed change and deciding whether

or not to merge a PR into a repository [26]. Previous studies of software engineering suggest that code reviews serve many purposes, such as identifying issues with proposed changes, collaborative problem-solving, clarifying, and encouraging knowledge sharing amongst team members [5, 18, 54]. In this thesis, we studied ∼7K references made in public GitHub PRs, analyzing what types of information objects are mentioned and how these references are expressed in PR discussions.

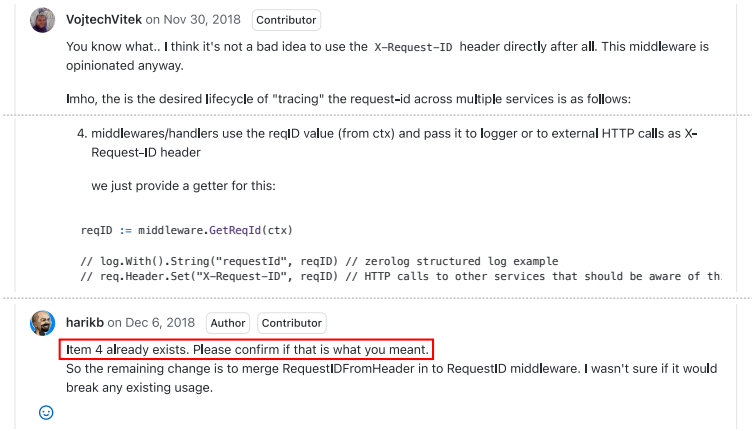

Figure 1.2: This figure shows the slice of discussion from a PR to showcase communication breakdown due to a wrong reference shown in red box. The grey dashed line indicates abridged content not shown in the capture for brevity

PR discussions frequently involve *references* to people and information objects, such as source code and documentation. These references are the focus of our work in this thesis. Theories of grounding in communication explain that referential behaviors are crucial for establishing shared understanding [14, 15]. As follows, two instances exemplify that referencing is a critical communication device for establishing a common ground, which can delay or impoverish the code review process upon failure. In the example shown in Figure 1.1 the submitter committed multiple code files and provided a vague description of the feature implemented in the PR without referencing the right files in the comment. So, the reviewer was confused which files to review and had to write a reply asking the submitter to provide an explicit reference ("point") to the file. The second example in Figure 1.2

shows that unclear references can cause delay or misunderstanding. The reviewer proposed an alternative API using the function name "getReqID()" which exists in the project, but the function had the same name as another function in an existing API. The homonym confused the PR author as expressed in their reply.

Previous studies have investigated referencing in educational discussion boards [70], remote collaborative work [23], and video-based comment threads [13, 64]. However, there has been limited work on understanding PR discussions [66, 68], thus, this thesis undertakes to fill this gap by describing software developers' referential behaviors. By filling this gap in the literature, we provide a clearer picture of PR reference behavior and pave the way for supporting PR discussions with better tools.

Referencing can impact the quality of PR discussion, and ultimately the maintainability of the project source code. To reach a decision on whether or not to merge a proposed change, references are crucial for creating a shared understanding of the consequences of accepting or rejecting a PR. In software engineering studies, there are many known factors that predict and impact PR outcomes [26, 27, 59, 60, 65]. For example, Zhang et al. [68] found that references to stakeholders through @-mentions impacts the amount of time that a PR will take to be processed. Identifying *what* and *how* people make references in PR discussions may lead to (1) better understanding of stakeholders' decision-making process, and (2) identifying the design requirements for better facilitating these discussions.

Referencing features are currently available in most online software development platforms, such as GitHub, GitLab[1], Gerrit[2], Bitbucket[3], and SourceForge[4]. The GitHub PR interface [3], for example, supports automatic referencing of platform-specific entities, such as commits, issues, PRs, and users, in addition to supporting direct URLs. However, the CSCW and HCI research literature has not extensively investigated the referential types and expressions that software developers make in PR discussions, so it is not known how well referencing is supported by these existing PR interfaces. In our analysis of PR discussions, we found that there are

[1] https://about.gitlab.com/
[2] https://www.gerritcodereview.com/
[3] https://bitbucket.org/product/
[4] https://sourceforge.net/

many types of references that are under-supported by GitHub, especially visual and interactive information objects, such as animations and UI elements. Beyond the context of software engineering, researchers have previously noted that features for referencing textual and visual information objects are necessary, but lacking [13, 23, 43, 64]. We also found that software developers use a wide range of expressions to reference information objects, hinting at a need for flexible communication in code reviews. In this thesis, we offer an empirical grounding of software developers' needs with respect to referencing in PR discussions. These insights can inform designs of referencing features that satisfy the needs of discussants in PR platforms and online communities.

We explore the following research questions:

1. What information objects do stakeholders refer to in PR discussions?
2. How are these references communicated?
3. What project attributes affect referencing behavior in code reviews and how?
4. What, if any, relationships exist between referencing and PR outcomes?

With these questions in mind, we conducted an analysis of a diverse sample of ∼7K references from 450 public GitHub PR discussion threads. We identified the references, including their type and expression, through a qualitative content analysis. We examined these referential behaviors using exploratory data analysis and hierarchical clustering on principal components. As a preview of our findings, we found that source code elements, such as variables and functions, are the most frequently referenced type of information; however, these did *not* make up the majority of all references. We found that the GitHub referencing interface has limited support for referencing source code. We also found that stakeholders have different referential practices, like pointing to a specific part of a documentation, such as READMEs as compared to external documentation. Finally, we found that some referent types, such as bots and tests, appear more often in accepted PRs than in open or closed PRs.

The contributions of this thesis are four-fold:

1. Two reference oriented taxonomies: one for what information is referenced (referent type) and how this is communicated (referential expressions).

2. Empirical findings from 450 PRs, identifying the empirical accounts of referencing support on GitHub; various referencing patterns which influenced by contextual factors that shape the use of references during discussion.

3. Design implications for PR discussion interfaces that are enhanced with type-based referencing based on our analysis, and,

4. An annotated dataset of $\sim$7K references, from 450 public PRs on GitHub.

Chapter 2

Related Work

The aim of this thesis is to understand the use of references in PR discussions. References in a discussion are useful in building the shared understanding among discussants and thereby improve communication. In software engineering, a discussion among developers during code review tends to include references to a wide variety of information entities. Therefore, providing references is important for progressing a PR discussion towards a resolution. In this chapter, we present a review of the past studies that highlighted the importance of references for building common ground in communication. We also present literature that discusses how a discussion serves as an important part of decision-making in the code review process along with other technical and social factors. In the later part of the chapter, we describe how the choice of research tools, such as taxonomy and dataset, have been employed in past research studies and how these are applicable in the context of our work. Finally, we give a brief overview of previous work designing better tools and interfaces to improve the code review communication process.

2.1 Common grounding through referencing

Grounding in communication is when people share knowledge and beliefs and is important for effective and efficient collaboration [15]. People frequently use references when communicating. A *reference* is a relationship between two objects, where one object connects or links to another object. Common ground allows dis-

cussants to identify and understand what is being referenced through their shared knowledge and beliefs [14]. For example, demonstrative references (this, these, that, those) with more than one potential referent can be dereferenced when common ground has been established [16]. Consequently, establishing referential identity, i.e., correctly identifying the object of a reference and the ability to dereference it, is a crucial aspect of communication [47].

Previous research has looked into referencing in different kinds of collaborative platforms [11, 13, 37, 64]. Chua et al. [13] developed an asynchronous discussion interface that supports complex non-verbal referencing to visual materials like documents and videos to support forum-style discussions in Massive Open Online Course environments by contextualizing the references in a thread. In a remote collaboration setting, AlphaRead [11] helps to make and resolve references to physical objects. Yarmand et al. [64] studied what types of information are referenced in YouTube comments, and how these referents are expressed. In addition, they showed how common referential behavior can be facilitated with an alternative commenting interface which improved engagement.

Our findings add to the existing CSCW and HCI literature, a better understanding of referencing behaviors specific to code review discussions and novel design implications for enhancing referencing to information resources in PRs.

2.2 Source code review

Effective communication has a significant impact on the software development life cycle. A quality modern code review process is important for ensuring the long-term maintainability of the code base [46]. Asynchronous code review is now supported by collaborative software development tools and platforms, meaning code feedback is often provided through PR-style discussion threads. Kononenko et al. [33] found that developers still expect asynchronous code reviews to be clear and thorough. Efstathiou et al. [22] evaluated the effectiveness of code review comments based on theories of rhetoric and discourse coherence by presenting sentence-level relationships within a comment. Similarly, Viviani et al. [63] investigated the linguistic representation of comments to study how design discussion is embedded in code reviews and found that software developers often use forms

like paragraphs to express it. While there is a growing body of research assessing the linguistic aspects of code review comments [20, 33, 63], little attention has been paid to the references used in the comments, which is a key for establishing a common ground [15].

We provide a detailed account of references developers make and how these references are expressed in their comments. We also provide qualitative descriptions of developer's referential practices which can inform design implications for building tools to support them.

2.3 Factors that affect PR outcomes

Given a PR, the reviewer's task is to decide whether or not to accept the proposed change. Discussion is often an important part of this evaluation process. Previous research has analyzed PR discussions to identify factors that impact PR merging [26, 27, 59, 60, 65]. Gousios et al. [26] found that PRs that propose changes to parts of the repository that are being actively developed are more likely to be accepted. In a follow-up study, Gousis et al. [27] conducted a large-scale survey of reviewers and found that the presence of tests, overall code quality, and the degree to which the code fits into a project's technical design all influence whether a PR is accepted.

In addition to these characteristics of the suggested source code, various social signals also play a role in whether or not a PR is merged. Tsay et al. [60] found that the strength of connection between a PR contributor and the stakeholders conducting the code review is important. Soares et al. [59] found that PR evaluators also take into account a PR contributor's reputation, their degree of contribution to the source code repository, and their experience (e.g., first PR vs long-time contributor). Together, these studies suggest that social factors can build trust in the proposed contribution.

The previous work has primarily relied on easily quantifiable variables to measure how much source code and social factors affect PR outcomes, such as the number of commits, the number of files changed, and the number of PR discussion comments [27, 59, 60, 65]. Less attention has been paid to analyzing the information entities that are referenced by participants in their PR discussions. Our research fills this gap by analyzing the information resources stakeholders refer-

ence in their PR discussions and how these references are correlated to the PR outcome.

2.4 Taxonomies as a tool

Analyzing comments in code review discussions is a growing area of research [8, 21, 39, 48, 54]. To better organize the use of comments in the PR evaluation process, a number of classification schemes have been created. For example, Pascarella et al. [54] provide a classification of the information needs of developers who review source code. Their taxonomy contains seven top-level categories and 18 sub-categories classifying needs like seeking rationale for developing correct understanding of the code, suggesting changes, and requesting additional actions. Ebert et al. [21] put forward a framework on the types of confusion that arises in code reviews by analysing software developers' comments. Both of these studies have focused on the cohesion of the overall discussion, and did not investigate developer's specific referential behaviors. Previous work in Information Science has investigated in studying people's referential behaviors and developed taxonomies to understand the types and forms of references [6, 34, 42, 64]. Yarmand et al. [64] is the closest to our work which classified the referent types and referential expressions of the references made in YouTube comments.

The research literature has had success creating taxonomies in order to describe the communicative behaviors of a wide range of communities which has also inspired the design choices of our taxonomy of references in PR discussions. For example, we chose inheritance based relationship as the classification scheme for our taxonomy, inspired by Ebert et al. [21] work which provided a taxonomy of 30 types of confusion by analyzing 307 review comments. Similarly, the choice of building two distinct taxonomies for types and expressions was inspired by the previous work by Yarmand et al. [64] which also gave two taxonomies by analyzing YouTube comments. Because an additional taxonomy of expressions helped in collecting the data and statistically analysing the referential usage patterns in depth. Thus, we developed two taxonomies derived from the data for this purpose: one for the information resources (referent types) that developers refer to in their discussion comments, second for the referential expressions (expression types) they use

to communicate these resources. These taxonomies provide the basis to explore the referential practices in PR discussions as described in Chapter 3.

2.5 GitHub dataset as a research contribution

GitHub is an online open-source software development platform. It hosts more than 100 million repositories with more than 56 million active developers [4]. GitHub's Pull request feature [1] integrates code reviews, discussions, bug tracking, and continuous integration loops within one interface which enhances the collaboration aspect of the platform even further. Therefore, GitHub has been the primary choice for data collection in many previous studies aimed at understanding pull request behaviors and open source software development in general [27, 32, 55, 65].

There have been significant GitHub dataset contributions in the past which have been extensively utilized by the research community. GitHub Archive Project[1] is the initiative started in 2011 to compile GitHub's data by capturing the public event timeline using GitHub Events API[2]. In 2013, Guosios et al. [24] published "GHTorrent" dataset which was similar to GitHub Archive but specifically designed for research purposes. Along with public events on GitHub, it also captured the content associated with each event exhaustively to provide a structured representation of the data which helps in querying the full history of projects and developer's actions on GitHub. In 2014, Guosios et al. [25] published a "pull-reqs" dataset of 350,000 pull requests curated from the GHTorrent, specifically designed to study the pull request development model on GitHub. All of these dataset provides an offline-mirror of GitHub's data for large-scale quantitative and qualitative research, none of them provided a structured representation of the information resources in the comments which can be utilized for understanding people's referential behaviors qualitatively or quantitatively.

In this research, we are contributing a dataset of $\sim$7k references identified in 450 public PRs on GitHub which is curated by manually reading each comment annotating references in it. This dataset presents an opportunity in analyzing peo-

[1] https://www.gharchive.org/

[2] https://docs.github.com/en/rest/reference/activity#events

ple's behaviors of referencing different information entities during a discussion.In addition, it can be used to dive deeper into understanding co-occurrence patterns of references and how other project attributes can affect referential behaviors. The details of the dataset are shared in Appendix A.

2.6 Designing code review interfaces

The past research in code review interfaces is mostly limited to enhancing the modality of interactions. There have been several code review tools such as Google's Mandarin, Facebook's Phabricator[3], CodeFlow[4], and Gerrit[5] which provide features to annotate the source code directly with comments and a chat interface to interact with others. GitHub's pull request features a similar discussion interface which can interleave code review comments with their discussion comments in a single thread. However, the major mode of discussion in all these review tools is primarily text-based. Few studies in the past extended this feature with the use of multi-modal interactions. Begel et. al [7] introduced the concept of documenting code comments by recording voice and embedding it in the source code. Hao et al. [29] extended the concept by making a multimedia commenting tool (MCT) which can record audio, video, and mouse interactions with the code and add it to the source code which can be replayed by code reviewers. This tool is integrated into the IDE which makes it useful not only for code reviewers but also for other developers to read the code. Chen et al. [12] introduced "Codeon", the asynchronous discussion interface which embeds code under discussion in the message to get assistance from remote developers. Park et al. [53] prototyped a tool to link online discussion conducted outside the programming environment within the IDE to the code it discusses which they called post-literate programming. Henley et al. [30] enhanced the existing code review interface integrated into an IDE by adding a visual representation of an automated code reviewer, CFar, which helps in identifying bugs in the code and leaves comments on the source code in a natural language like any other user. This automated bot is considered similar to other reviewers in the system. In general, the past research improved the way code reviews

[3] https://secure.phabricator.com/
[4] https://www.getcodeflow.com/
[5] https://www.gerritcodereview.com/.

are recorded and presented by exploring different modalities.

In our research, we are focusing on the content people mention in their comments in PRs which can lead to designing better interface support for code reviews. By identifying people's referential behaviors in their comments, we shared design implications in Chapter 5 which satisfy the communicative needs of the PR authors and reviewers.

2.7 Summary

Previous work suggests that common grounding is an important aspect in communication and enables discussants to refer to objects and understand those references. In the software engineering context, code review is an important process that is frequently conducted asynchronously in text-based discussion threads. Previous work has identified a number of attributes — both source code-based and interpersonal — that affect whether or not a proposed change is merged into a source code repository. This thesis builds on this work by examining the references software developers make in PR discussion, and how those references relate to the PR decision-making process. Further, aligned with previous work, we create two taxonomies to provide a frame to organize the types and patterns of references in PR.

Chapter 3

Methods

To study the referential practices in PR discussions, we represented the *types* of references that software developers use in PR discussions, and how those references are *expressed*, by creating taxonomies of *referent types* (Table 3.2) and *referential expressions* (Table 3.3) using the research design presented in Figure 3.1. These lay the foundation for analysing referential behaviors in Section 3.3.2.

To create these taxonomies, we conducted a qualitative content analysis (QCA) [35] on ∼7K references in ∼2K comments from 450 public GitHub PRs. QCA is a bottom-up process that identifies themes through analysis of the data [67], and is a useful method for generating inductive taxonomies [28, 41, 42]. For data collection, we used GitHub because it is the largest platform for open source software

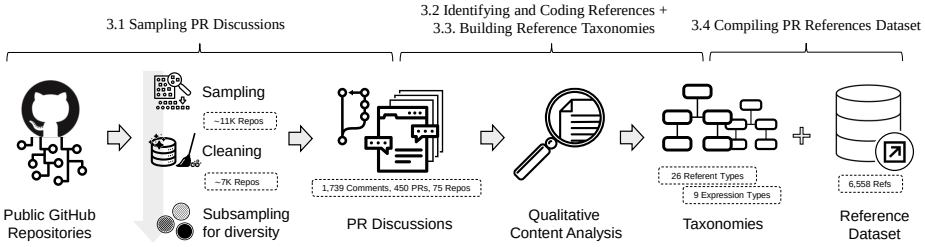

Figure 3.1: A flow diagram representing the research design we used to compile an annotated dataset of ∼7K references from 450 public PRs from GitHub. The dataset is used to analyze referential patterns in PR discussions in Section 3.3.2.

development. GitHub's PR feature is similar to those provided by other platforms, such as Bitbucket, Gerrit, and SourceForge. We conducted the QCA in three steps:

1. *Sampled PR discussions*: we sampled 450 PRs from GitHub, containing 1,739 comments.

2. *Identified and coded references*: we analyzed each PR thread to identify and code references.

3. *Built the reference taxonomies*: we used affinity diagramming to categorize the references into two taxonomies: 1) referent types and 2) expression types.

3.1 Sampling PR discussions

We collected a stratified sample of 450 public PR discussion threads from GitHub. Our sampling strategy involved five steps. First, we scraped top 10,786 repositories by stars count using GitHub GraphQL API[1]. This is because PR threads are organised into huge collection of repositories on GitHub. We chose number of stars of a repository as the selection criteria because it has been a popular choice by researchers in many previous work for for empirical studies [9, 10, 52, 56, 58].

In the second step, we filtered out repositories in the following order:

1. We removed 823 repositories which were non-software repositories. We used primary programming language information associated with each repository to distinguish a software repository from non-software repository like books, documentation etc.

2. We removed 3,146 repositories which did not have a sufficient number of PRs (minimum 2) in each of open, merged and closed state as required for sampling.

3. From the remaining set, we removed 50 more repositories which had no description or keywords associated with it which is required for topic modelling in the next step.

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

Table 3.1: Diversity criteria for sampling repositories

Criteria	Description
Topic Composition	A 29-dimensional vector for each repository output by topic modelling represents the probability distribution of the repository according to 29 topics.
Number of Stars	The number of times the repository has been starred.
Number of Contributors	The number of contributors who submitted code commits in the repository.
Number of PRs	The number of Pull requests that have been submitted to the project.
Main Language	The main programming language used in the source code of the repository identified by the GitHub platform.

The filtering left us with 6,767 repositories.

In the third step, we identified the application domain of these repositories using the Latent Dirichlet Allocation topic modelling. Application domain is the primary diversity criteria for sampling PRs for our study to balance out the domain specific concerns in our collected samples. Topic models that we used are the family of algorithms which extract topics from unstructured text. These algorithms do not infer the meaning of the words in the text, instead they identify the list of words that occur in statistically meaningful ways and cluster them into different topics. Using description and keywords associated with each repository, we represented them as a corpus of text documents and ran the LDA algorithm multiple times with number of topics parameter ranging from [5,50] with a step size 3 i.e., $\{5,8,11,....,50\}$. The number of iterations parameter was set to 1000, and built-in hyper-parameter optimization was enabled with 'optimize-interval' parameter set to 10. For other parameters, we used default values set in MALLET[2] library. In total, we ran topic modelling 13 times on these repositories. The output of each execution was a set of topics represented by top ranked list of keywords and a topic composition (i.e., the probability distribution) of these topics for each repositories

[2]http://mallet.cs.umass.edu/topics.php

in our collection. To identify the right number of topics for our repositories, we manually analysed each execution result to find one which has maximum number of coherent topics. Therefore, we selected 29 topics to be an optimum number to categorize the repositories collection. The list of 29 topics identified by topic modeling is provided in Appendix D.

In the fourth step, we used Nagappan et al.'s [49] *sample coverage* measure to create a representative sample of 75 distinct repositories from the collection of 6,767 repositories, accounting for the five diversity criteria as mentioned in Table 3.1. Our sample of 75 repositories covered 40.8% of the total variance.

In the final step, we randomly sampled 450 PRs from 75 repositories. To this end, we downloaded 21,786 PRs from 75 repositories and removed 1,805 PRs without comments. From remaining collection of 19,981 PRs, we randomly sampled two merged, two closed, and two open PRs from each of the 75 repositories, resulting into the dataset of 450 PRs (2 PRs $\times$ 3 Statuses $\times$ 75 Repos = 450 PRs). The final PR dataset contained 2,189 comments including 1,739 threaded messages and 450 PR titles. This sampling pipeline was executed on 18 November, 2019.

3.2 Identifying and coding references

We identified and coded all the references within the 450 PRs, and classified the *referent* and *expression* types. A *reference* is a word or phrase used in a PR discussion comment to signal an information object. A reference is comprised of two components:

- **Referent**: A referent is an information resource that is mentioned by a PR discussant. For example, a referent could be a variable, function, file, commit, documentation, URL, etc. It could also be an abstract concept (for example, JSON format, async tests, etc). These can be singular, plural, or a collective.

- **Expression**: A referential expression is how the information resource is encoded in discussion, for example, "@reisenberger", "#236", and "Reading label is not responsive". Here a name, id, or description is used to encode the referent.

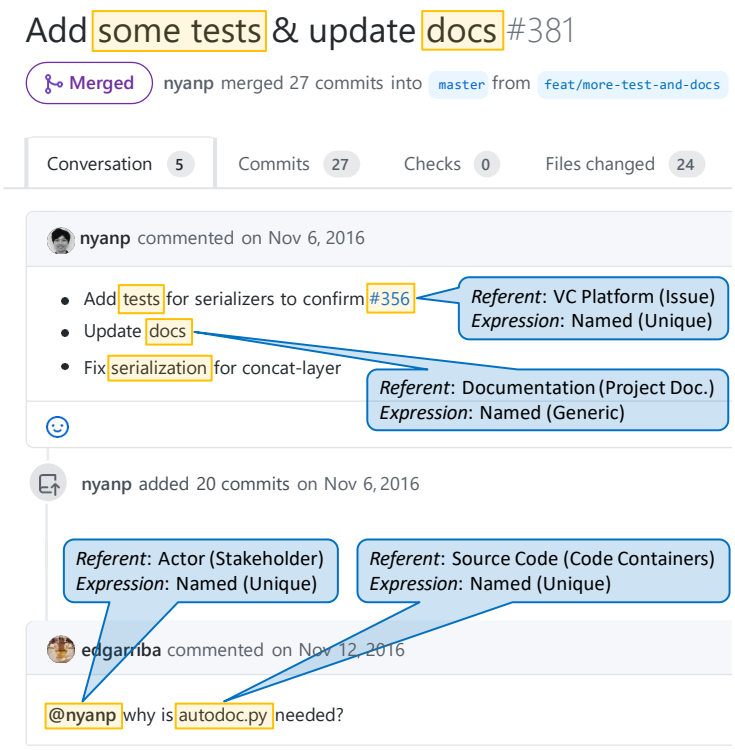

Figure 3.2: An illustrated example of coded references. We assigned two codes to each reference. For each reference (highlighted in a yellow box), the type (i.e., top-category) and sub-category of the referent and expression are indicated as "Type (Sub-category)" in a blue box. See Table 3.2 and Table 3.3 for the full list of referents and expressions.

Two of the project collaborators performed QCA, and the process was discussed among all collaborators to resolve conflicts. To identify references, we looked into the words or concrete phrases used to reference an information object, and then coded the reference's referent as well as the expression used to encode the object. In total, we identified 6,558 references, which we analyzed in Chapter 4.

The coders worked together on 25% of the dataset for training. During the training phase, a code book was co-authored by the coders in three iterations. We developed codes for both referent type and referential expression until we reached

saturation, and refined the codes during this phase by discussing with all members of the research team. The final code book contains 196 codes for referent types and 33 codes for expression types. An illustrated example of coded references is shown in Figure 3.2 where each reference is highlighted and is assigned two codes, one for referent type, one for expression. It took a total of 560 hours per coder to code 450 PR discussion threads in which the training phase took 40% of the time. The code books for referent type and expression types are available in Appendix B and Appendix C respectively. We used NVivo for this analysis.

Once the code book was established, the coders independently coded 74% of the PRs (N = 334). We tested inter-coder reliability on 482 references identified in these 334 PRs. The Cohen's Kappa scores [17] were 0.76 for the Referent Taxonomy and 0.72 for the Expression Taxonomy. These scores indicate a strong agreement between the coders [45]. The disagreement instances were categorized into 12 notable cases and resolved after an hour-long discussion between the two coders.

3.3 Building reference taxonomies

We iteratively built and refined our taxonomy to classify 196 referent codes identified during the previous step into 6 top-level categories and 26 sub-categories that make up the referent taxonomy (Table 3.2). The process of building taxonomies took 400 hours in total over four iterations by the lead collaborator who discussed with all other project collaborators for evaluation and refinement. To make the classification generalizable, we organized these codes by the inheritance relationship. Codes which exhibit an *is-a* relationship are grouped under the same category. For example, a variable, function, class *is-a* SOURCE CODE; a user, organisation, bot *is-an* ACTOR. The SOURCE CODE category in the taxonomy consists of a variety of source code information objects, including variables, functions, classes, libraries, packages, and so on. The other categories in the taxonomy such as ACTOR, VC PLATFORM, COMPILATION AND EXECUTION RESULT, DOCUMENTATION, and DEV TOOLS AND ENVIRONMENT collectively contain non-source code entities. In the last iteration, we added few additional new codes generated during the last batch of coding in the final taxonomy. Similarly, 33 expression codes

Table 3.2: Taxonomy of referent types, with examples.

Type	Sub-category	Definition	Example(s) from Dataset
SOURCE CODE	*Code Containers*	A container unit which contains the source code, typically a file.	Only change in abstractclientbase.js is this...
	Code Elements	The building blocks that make up the source code, such as symbols, keywords and various constructs.	.doesNotThrow() is currently not matching the error message...
	Inputs and Values	Any literal value or data specified in the source code	If you accidentally pass a None or False value, it will return without check.
	Code Libraries	A collection of non-volatile resources used by the source code.	In the development of a script to periodically send statistics of performance of three.js examples...
	Test	A source code written specifically for testing purposes.	The build is failing because we need to update the snapshot test for this component.
	API	A source code written to be used by other code or project.	We are trying to keep the surface API as consistent as possible
	Assets	Other artifacts in the project which accompanies the source code.	Updated AWS icons.
ACTOR	*Stakeholder*	People or groups affected by the software project lifecycle.	@reisenberger thanks for the tag!
	Agent	A software program that acts for a user in PR discussion to automate tasks.	And Travis CI broke for unrelated issues that I need to investigate.
	Organisation	A larger body of people who have stakes in the software project.	Pending assignment with my open source review board internally. I'll follow up.
VC PLATFORM	*Version-Control Entities*	Entities related to the version control system of the software like branches, hooks, merge conflicts etc.	Otherwise this branch is a great starting point for you to work on the update.
	Repo	A global container of project related resources and artifacts.	@bordeo this repo is actually dead.
	Issue	A bug or feature raised by stakeholders related to the software project.	A possible fix of the issue #48
	PR	Code submission request raised by contributors in the project.	I like the simplicity of this solution, which also has no executional overhead versus PR #248...
	Comment	The information expressed by stakeholders in the discussion of the PR.	I am still thinking about the last comment regarding async behavior though.
DEV TOOLS AND ENVIRONMENT	*Software Development Tools*	Tools used by people to develop the software project.	This enables some new warnings, as recommended by Xcode.
	Automation Tools	Tools used to automate lifecycle tasks of the software development process.	In the last commit, I upgrade Gradle to the 3.0.0-beta2...
	Programming Language	The language used to express the source code.	I know that semicolons are optional in JS.
	Environment	A virtual environment where the software is Executed.	This is very noticeable on the iPhone simulator with slow animations enabled.
	Platform	An environment where the software is executed.	Seems it's only broken on android btw, iOS works correctly.
COMPILATION AND EXECUTION RESULT	Warning	Messages that indicate some issue with source code after compilation or execution, without halting the activity.	fix compilation warnings on older Python versions
	Error	A result obtained as an outcome of unsuccessful compilation or execution of the source code.	Also, I wasn't able to run gradlew build, it gave me some Javadoc errors which I'm not sure how to fix.
	Output	A result obtained as an outcome of successful execution of the source code.	Made the following changes to make logs look more like Eclipse/Studio logcat...
	Application	A runtime instance of the source code after successful execution.	The app seemed to behave the same before/after this change.
DOCUMENTATION	*Client Documentation*	End-user documentation of a project.	If this is the most important factor, it would be great to have the README reflect it then.
	Project Documentation	Project development lifecycle related documents used by stakeholders.	Please refer to our pull request process documentation to help your PR have a smooth ride to approval.
	Reference Documentation	Third Party documentation which are referred by stakeholders to conduct their activities.	Extend the openapi validator to allow the examples field, see https://swagger.io/docs/

Table 3.3: Taxonomy of expression types, with examples.

Type	Sub-category	Definition	Example(s) from Dataset
NAMED	*Unique/Distinct*	A unique attribute of the referent	ReflectUtils has an overload that takes the cxtor...
	Generic	General/collective word or phrase to identify referent	Let me know why you created this pull request, closing it for now.
CONTENT-BASED	*Descriptive*	An expression which serves to describe the referent in text.	Now they're colored too (if colored nicknames option is enabled).
	Verbatim	An expression which uses the exact same representation of the referent's content.	Below that you can add stats.fps = (frames * 1000) / (time - prevTime);
LOCATIVE	*Absolute*	An expression which uses the exact location of a referent.	The loop function in src/native/utils/game-loop.js has been changed from arrow function to regular function...
	Relative	An expression which uses the location of a referent relative to the current location where the expression is used.	@smurching Never mind my comment above.
TEMPORAL	*Absolute*	An exact point in time	Thanks @filbertteo ! Released in react-game-kit@1.0.6
	Relative	Relative to current point of time	Thanks for your contribution! We will work on releasing new version soon.
	Range	Interval of time	The change has to be made in a way that supports both Butterknife 7 and older versions.

were classified into 4 top-level categories and 9 sub-categories in our expression taxonomy (Table 3.3). The full directory of referent and expression codes of these taxonomies is provided in Appendix B and Appendix C respectively.

3.3.1 Compiling PR references dataset

We have compiled a dataset of ∼7K coded references from 450 PR discussion threads in CSV format and have made it available publicly. The schema of the dataset is available in Appendix A. Each row in the dataset represents a reference identified from a PR and is described by 26 columns of information related to the repository, PR, and the assigned referent and expression codes, categories and sub-categories. Also, each reference contains the comment from which it was extracted, marked with the location of the reference within the comment. It took 360 hours in total to develop, test and execute the data extraction pipeline to compile the dataset from the coded PR threads. We used this dataset for our exploratory data analysis of referential behaviors, which we present in Chapter 4.

3.3.2 Analysing PR references

We used the dataset to identify referential behaviors in PR discussions, and how these behaviors are impacted. Prior work has suggested various factors which impacts the PR evaluation process [26, 27, 59, 60, 65]. Building on this work, we

formulated three guiding principles that we applied in analyzing referential behaviors in PR discussions:

1. We determine the commonly referenced referent types and expressions and their relationships.

2. We consider how project attributes like application domain, programming language impact the use of references in the discussion.

3. We also investigate the relationship between referent types and the PR outcome.

To this end, two of the project collaborators conducted an exploratory data analysis [62] to identify patterns in the references dataset. We used a mixed approach which includes inductive and deductive inquiry of the data. Using the guiding principles mentioned above, we plotted frequency distribution of the references and generated 34 distinct patterns. And then, we performed qualitative reading of the samples in PR discussions to confirm the relevance of these patterns. These patterns were discussed with four other project collaborators to reach an agreement. The two collaborators spent around 400 hours in total in analyzing the PR references.

To explore the co-occurrence relationships between the referent types, we performed cluster analysis. We created a matrix of PRs by the counts of each referent type in the PR with 450 rows, one for each PR, and six columns, one for each referent type. We used principal component methods, specifically correspondence analysis [61], as a preprocess for clustering. We then used hierarchical clustering on principal components to identify the relationships between the top-level referent types. We used the single linkage method, which calculates the minimum distance between a pair of observations, to cluster through a bottom-up, agglomerative approach. The resulting dendrogram (Figure 4.3) was cut at the partition with the higher relative loss of variance, which yielded three clusters. We plotted these clusters on the factor map provided by the correspondence analysis, as suggested by [31] (see Figure 4.4). The relevant analysis of co-occurring referent types is presented in Chapter 4.

Chapter 4

Findings

Overall, referencing information resources is prevalent in PR discussions. There are 6,558 references in 2,189 comments. 93.5% of the comments referred to at least one information entity or more. On average, a comment has 2.99 references (SD = 3.15 , Min = 0, Max = 59). Here, when we say "comments" they include not only threaded messages of the PRs but also their titles. A PR, on average, has 4.86 comments (SD = 4.33, Min = 2, Max = 54).

Referencing is critical communication device for establishing a common ground. As shown in Figure 1.1 and Figure 1.2, discussants in a PR misunderstood another or got confused about other's comments due to the breakdown of reference to information resources. Therefore, having correct and unambiguous references are also crucial for successful code review discussion.

The following sections document our findings on how well the current interfaces support referencing in PR, how people use references, and how referencing patterns might indicate different PR outcomes.

4.1 Empirical Accounts of Referencing Support on GitHub

PR discussants in GitHub referred to information resources in three ways. GitHub's auto-link feature [3] detects a reference prefixed with a special character (e.g., "#456", "@reisenberger") and automatically converts it into a link. However, auto-

Table 4.1: Distribution of all references by referent type (N = 6,558)

Referent Type	**Total Count of References**	**GitHub Auto-link and URL-link References**
SOURCE CODE	2986 (45.5%)	76 (1.2%)
VC PLATFORM	1733 (26.4%)	424 (6.5%)
ACTOR	607 (9.3%)	362 (5.5%)
DOCUMENTATION	544 (8.3%)	156 (2.4%)
DEV TOOLS AND ENVIRONMENT	364 (5.6%)	4 (<0.1%)
COMPILATION AND EXECUTION RESULT	324 (4.9%)	11 (<0.1%)
Total	6,558 (100%)	1,033 (15.8%)

linking currently works for a handful of referent types presented in our referent taxonomy such as VC PLATFORM (*issues*, *PRs*, and *commits*), and ACTOR (*stakeholders*). GitHub's interface can also detect URL strings in comments and convert them into hyperlinks, which we refer to as URL-link, to differentiate it from auto-links. For information entities that auto-links don't support nor are URLs, writing plain text description of the reference is the only option (e.g., "paperFontButton", "flake8 issues", "IE 11"). In this section, we focus on providing empirical accounts of to which extent the existing interface features support different types of references or leave certain types unsupported.

Among all reference instances in our dataset, the majority was in plain text while only 15.8% were either auto-links or URL links (Table 4.1). The plain text references were overrepresented, because, when referring to a collective class of referents or an established reference that appeared early in the thread, discussants tend to use *generic* terms such as pronouns, adjectives, common nouns, and collective nouns (for e.g., "issues", "unit tests", '"this PR", "the function", "it", etc.). These references do not establish any referential identity; they assume that the referential identity of the information resource is already established. To count direct, specific references only, we removed 1,904 of these second-degree references that are expressed in the generic terms (i.e., NAMED-*Generic* types in our expression taxonomy Table 3.3). In the remaining 4,654 references the proportion of auto-links and URL links went up as shown in Table 4.2.

Table 4.2: Distribution of non-generic references created using different referencing mechanisms (N = 4,654)

Referent Type	Auto-linked References	URL linked References	Plain Text References
SOURCE CODE	0	76 (3.3%)	2203 (96.7%)
VC PLATFORM	368 (41.5%)	56 (6.3%)	461 (52.1%)
ACTOR	332 (62.9%)	30 (5.7%)	166 (31.4%)
DOCUMENTATION	0	156 (38.5%)	249 (61.5%)
DEV TOOLS AND ENVIRONMENT	0	4 (1.2%)	318 (98.8%)
COMPILATION AND EXECUTION RESULT	0	11 (4.7%)	224 (95.7%)
Total	700 (15%)	333 (7.2%)	3621 (77.8%)

4.1.1 How well GitHub's referencing feature is being used

GitHub's auto-linking and URL links were popular modes of referencing when the interface supported the type of referents. The majority of ACTOR (62.9%) and a significant portion of VC PLATFORM (41.5%) were auto-linked references. URL linking was a popular way to link DOCUMENTATION. However, plain text took the majority in most of the other referent types, especially dominating SOURCE CODE (96.7%), DEV TOOLS AND ENVIRONMENT (98.8%), and COMPILATION AND EXECUTION RESULT (95.7%).

Auto-linked referencing feature is most frequently used for platform-related referent types

We found that GitHub platform's resources such as *Issue*, *PR*, and *Stakeholder* are the most frequently referenced types using auto-linked referencing interface (see Figure 4.1). These information resources are native to the GitHub platform and have built-in support by the auto-linking mechanism in the interface. For example, a developer can auto-link an issue or a PR by writing a prefix "#", as in "This fixes #9." Similarly, a user can be referenced in the comment by writing a prefix "@" with username as in "@reisenberger thanks for the tag!". The prevalent use of these references in comments is likely due to the auto-complete which fires up as soon as a user types the special characters (#, @) within the authoring interface which makes it easy to search and link references while commenting.

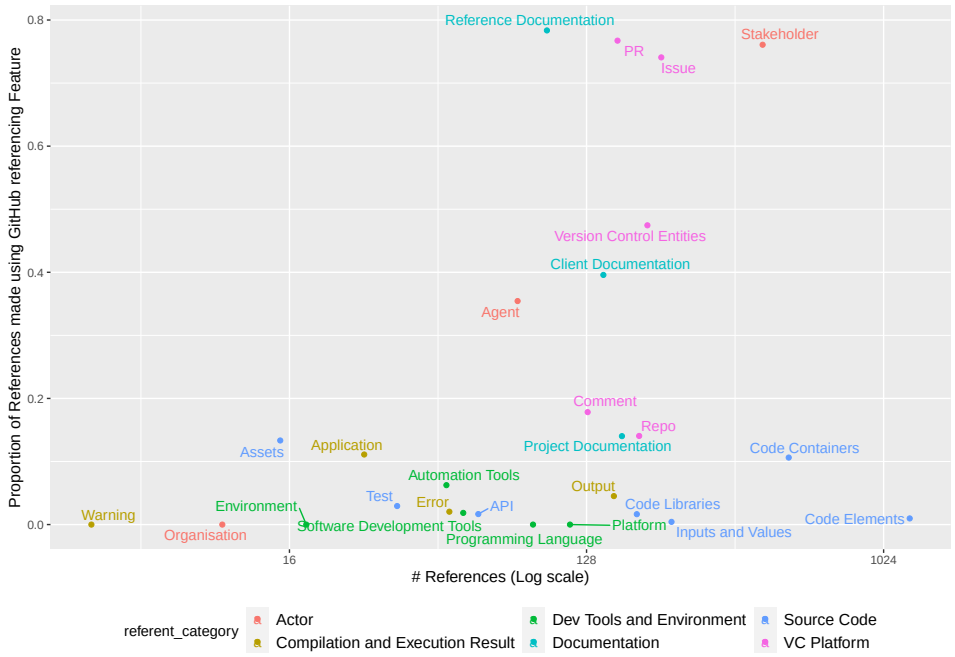

Figure 4.1: Distribution of referent sub-categories based on the number of non-generic references (x-axis) and the proportion of those references created using the GitHub feature (y-axis).

At present auto-linked referencing feature caters only to platform-specific entities, however, it does not provide full coverage. *Repositories* and Issue/PR *comments*, for example, are currently outside its support. We found that discussants use plain text referencing for 80% of references to other repositories on GitHub such as dependent library projects and forked repositories. For example, in one of the PRs, the submitter referenced another repository in his comment by linking URL as in "I made this from combining parts of the ipython/ipython and jupyter/jupyter Dockerfiles".

Linking standard URLs in comments is sparsely used by developers

GitHub's interface also supports creating hyperlinks to URLs. When a user provides a URL in the comment text, it renders into a clickable link. We found that 7.2% of references made on GitHub are URLs (Table 4.2). *Reference Documenta-*

tion which are outside GitHub such as external web pages and forum threads are majorly referenced by auto-linking URLs (see Reference Documentation in Figure 4.1). However, other types such as *Agents*, *Client Documentation*, and *Project Documentation* used it sparsely. A possible reason for sparse usage of linking URLs is because it is time-consuming for discussants to locate, copy, and paste the URL of the resource in the comment text. It is also sometimes an overkill for resources which are intrinsic to the project such as project docs, readme, bots, CI services, code files, therefore, these are primarily mentioned in plain text.

4.1.2 Majority of referent types are expressed in plain text

We found that the majority of references (77.8%) were written in plain text with no link (Table 4.2). The higher number of plain text references in PR discussion is mainly because of the lack of support of many referent types through GitHub's auto-linked referencing interface. Therefore, plain text is employed as the fallback option. For example, SOURCE CODE, which is a highly discussed information in PRs, has no support on the referencing interface and is majorly discussed in plain text (Table 4.2). Since plain text lacks the ability to link to the referent, it takes longer to express the referent information, for example, a reviewer has to explain the variable value in a long sentence as "The fontSize for paperFontButton is 14". Similarly, a PR in UI project has a comment as "Made box shadow color of white color option a darker shade of gray for better visibility" in which properties and values in the source code is explained in long text. It becomes even more challenging when the discussion involves *Application* referents such as runtime instances of source code as in this comment "Click on menu1, then menu2, then menu1 again → JS error is occurred." Here, the UI navigation is described in writing to reproduce the issue which is difficult to dereference.

On the other hand, the plain text is an appropriate choice of expression for some referent types such as DEV TOOLS AND ENVIRONMENT, COMPILATION AND EXECUTION RESULT, and DOCUMENTATION. For example, *Software Development Tools* such as automation tools, code editors, browsers, execution platforms are mainly presented in text as in this PR comment "Fixes issue with m.route in IE 11, caused by this line. history.state can be null and the Object.assign polyfill does not

handle it." This is because references to tools like "IE 11", "chrome", "windows", "android", "JVM", etc. are commonly used jargons in software engineering which appear quite frequently in discussion and can be easily understood in plain text.

We found that ∼19% of plain text references express the referents verbatim. For example, compilation/runtime *Errors*, and console *Outputs* are generally provided as-is by copying the output log in the comments (Figure 4.1). This is likely because *Errors* cannot be easily understood by other reference handles such as error codes only, therefore, the whole stack trace logs are useful in conveying the error in the discussion. However, producing verbatim text references by copying and pasting is time-consuming. Also, referencing and dereferencing a part of the verbatim text in subsequent comments also takes effort in writing to point to it.

On the other hand, some plain text references use descriptive phrases to reference information such as *API*, *Warnings*, and *Test Cases* (Figure 4.1). For example, in one of the PR comment, the reviewer suggested a new API as "it would be great to have a "should retry" callback somewhere that will decide if the retry is still relevant at the time it can execute", which was picked up in the thread and used as a reference handle while discussing the use case scenario by other discussants. Similar practices have also been observed with *Tests*, *Warnings* and *Issues* which are given short descriptive titles (for e.g., "coroutine test", "peer dependency warnings", "flake8 issues", "typescript bug", etc.) which served as the reference handle for discussion in the thread.

4.1.3 Source code is frequently discussed in PR discussions, but not supported by auto-linking

There are many references to source code in PR discussion despite these not being supported by the interface. We found that 45.5% of total references in our dataset are to SOURCE CODE referent types (Table 4.1). *Code Elements* and *Code Containers* are the most prevalent of these (54.9% and 22.5% respectively, see Figure 4.2). However, only 3.3% of total SOURCE CODE references were created using GitHub referencing interface, primarily by linking URLs (Table 4.2). Here we present referencing patterns in the two major source code types below.

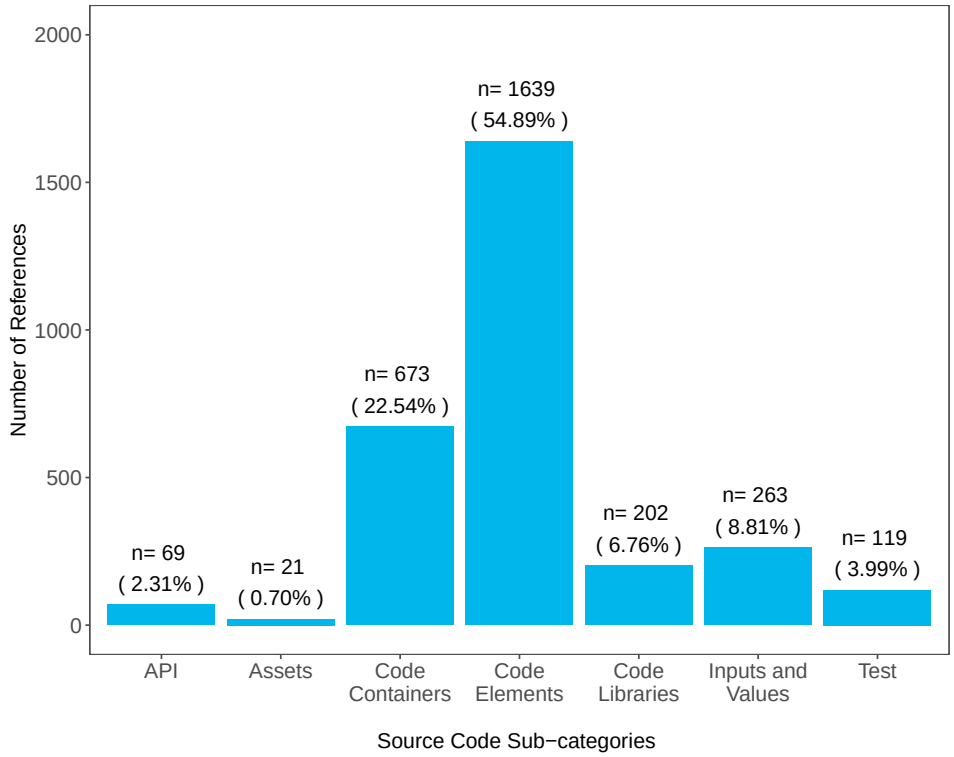

Figure 4.2: Distribution of SOURCE CODE Type references (N= 2,986 as per Table 4.1) by their sub-categories.

Plain text-based referencing to Code Elements can be time-consuming.

Code Elements (for e.g., variables, functions, classes, etc.) is the most frequently referenced (54.9%) SOURCE CODE type in PR discussions (Figure 4.2). 99% of *Code Elements* references are made using plain text in comments. Since there is no direct URL for code constructs like variables, functions, and classes, they are referred manually Table 4.3. For example, a PR contributor describes the content of the PR by referencing functions implemented in the class as plain text in this comment "Implements a new tracing support for uvloop adding the pair methods start_tracing and stop_tracing that allows the user to start or stop the tracing.". Further, the contributor also references classes in the PR discourse as "MetricsSpan, CounterSpan and TimingSpan controversial, this PR only implements a generic

Table 4.3: Distribution of references to Source Code sub-categories by different referencing mechanisms (N = 2,279)

Sub-category	Auto-linked References	URL-linked References	Plain Text References
API	0	1 (1.6%)	59 (98.3%)
Assets	0	2 (13.3%)	13 (86.7%)
Code Containers	0	56 (10.6%)	471 (89.4%)
Code Elements	0	12 (1.0%)	1217 (99%)
Code Libraries	0	3 (1.6%)	179 (98.3%)
Inputs and Values	0	1 (<1%)	231 (99.5%)
Test	0	1 (2.9%)	33 (97.0%)
Total	0	76 (3.3%)	2203 (96.7%)

Span that is a timing one". Referencing these resources becomes challenging in large repositories containing classes and functions with same names located into different files and folders. Often times developers try to resolve this ambiguity by mentioning the fully qualified name of the class in text as "when internally a new span is created using the TracedContext.start_span under the hood the parent span is passed to the Tracer.create_span, so allowing to the tracer to fetch some context information from this", which is time-consuming and error-prone. Sometimes, developers workaround this limitation by inserting the URLs for lines of code where the function or class is located in the file but it involves context switching by moving the developer away from the discussion interface. Also, dereferencing source code references in plain text is tedious for readers of the PR and requires a bit of switching to different web pages to find the source file and location where the referent is situated, in order to follow the discourse.

Developers can reference code containers in PR discussions by linking URLs but it is rarely used.

Code Containers, such as code files, folders, and lines of code, are the second-most referenced type (22.5%) of SOURCE CODE (Figure 4.2). GitHub interface allows developers to reference lines of code and code files by inserting URLs in the comments. However, we found that the majority of *Code Containers* references

Table 4.4: Distribution of PRs by SOURCE CODE and non-source code referents.

Referent Type(s)	Count of PRs
SOURCE CODE only	34 (7.5%)
SOURCE CODE and Non-source code	356 (79.1%)
Non-source code only	60 (13.3%)
Total	450 (100%)

(89.4%) are made in plain text while only 56 out of 527 (10.6%) were created using URL-links Table 4.3. For example, a discussant mentioned a line of code by specifying the line numbers in text form as in "Can we wrap the try started on line 97 in a single try/catch". Similarly for the code files as in "It would be helpful to have ICombinedSimpleModel interface be in the main.ts". This is possibly because typing the name of the file, or the line number is easier and quicker for comment authors than manually searching and inserting the URL. However, it is burdensome for readers to dereference these resources to build a common understanding of the context.

4.2 Contextual factors that shape the reference use

PR discussions is an activity to evaluate submitted changes to the code repository to make a decision to merge or reject. During the discourse, developers make references to various information resources relevant to the context of the discussion. These references are indexical in nature which are generally understood by knowing the associated context attached to them. The context also shapes how these references are made. Traditionally, a context is not readily considered as a static entity; rather, context is constructed and reconstructed through interaction during discourse. Courtright et al. [19] published several factors that play a key role in shaping the context in information practices which are applicable in PR discussions setting as well. In this chapter we present a qualitative description of how the use of references are shaped by these different contextual factors.

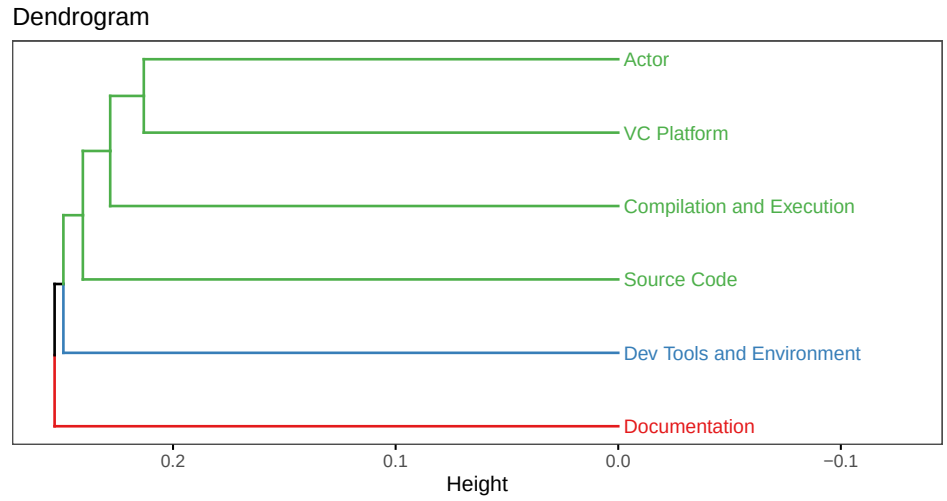

Figure 4.3: Dendrogram on principal components on the top-level referent types generated by the correspondence analysis and hierarchical clustering.

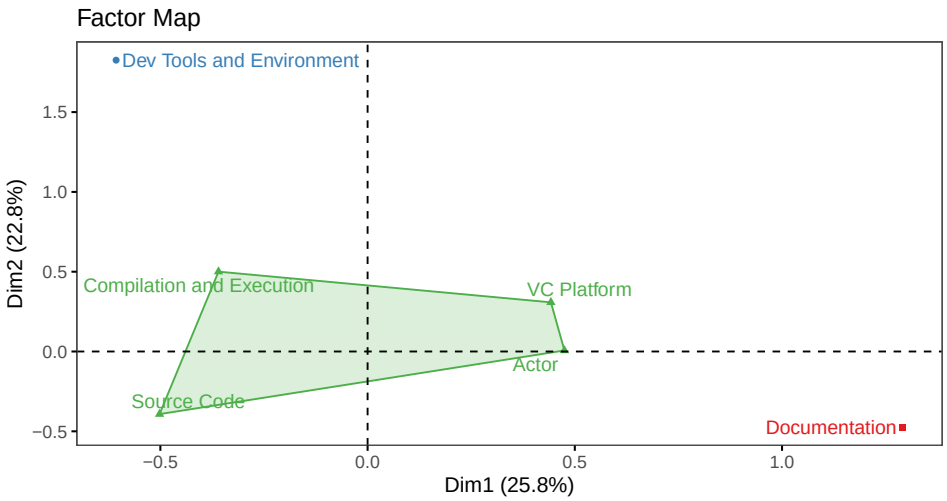

Figure 4.4: Factor map on principal components on the top-level referent types generated by the correspondence analysis and hierarchical clustering

4.2.1 Information Type

The type of information resource being referenced influences the use of references in the PR discussion. Some referent types can affect how a reference is expressed. Whereas, some referent types affect the use of other referent types in the discourse. For instance, we found that source code types frequently co-occur with non-source code types (Table 4.4). The cluster analysis shows that the source code references specifically co-occur with the ACTOR, VC PLATFORM, and COMPILATION AND EXECUTION RESULT referent types, but tend not to with DEV TOOLS AND ENVIRONMENT, and DOCUMENTATION (see Figure 4.4). Our detailed findings on how information type shapes the reference usage in a discussion are presented below.

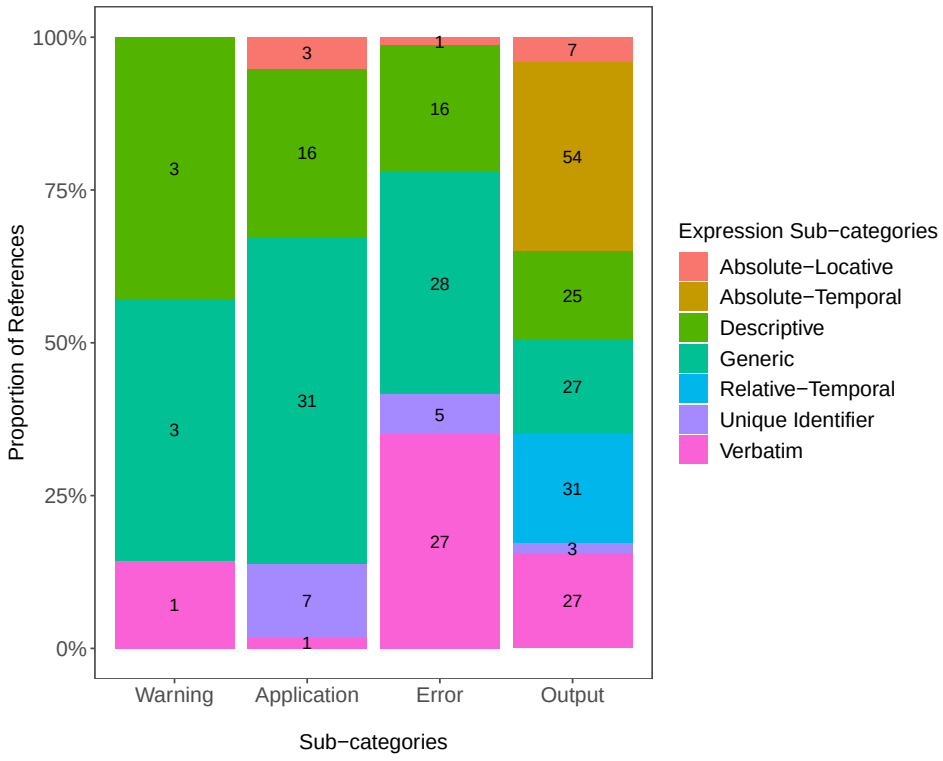

Figure 4.5: Distribution of Expression Sub-categories for COMPILATION AND EXECUTION RESULT.

Developers tend to express compilation errors verbatim and warnings descriptively

COMPILATION AND EXECUTION RESULT referent types such as *Errors* are referenced verbatim, either by copying and pasting the error text or with a screenshot image. *Warnings*, on the other hand, are expressed using descriptive titles or names only. From all the references to errors in our dataset, 34% are expressed verbatim. Similarly, 42% of references to warning messages are descriptively mentioned (Figure 4.5). In software development, errors are more technical in nature and can block the software development and execution activities as compared to warnings. Therefore, to diagnose the problem, one needs complete error information (e.g., a stack trace). This is why verbatim expressions are preferred for errors.

PRs that discuss source code mostly contain references to non-source code references such as issues and other PRs

81% of SOURCE CODE discussions contain at least one reference to VC PLATFORM types, such as *Issues* and *PRs*. Issues on GitHub also have a discussion thread of their own. Therefore, referencing issues in the PR discussion provides evidence of the rationale behind the decisions made by the contributor in the proposed change. Reviewers also often reference related or similar PRs in the discussion which is impacted by the current change to fast-forward the evaluation process, for example, "A possible fix of the issue #48 and Fixes #425". Also, in our qualitative exploration, we found that these references appear in different places in the PR. For example, some contributors mention them in the PR description, but other contributors or reviewers bring it into the discussion during the review process. Referencing issues and PRs not only helps in the evaluation process of the PR, but also helps others to track similar efforts, either in the space of issues, PRs, or commits.

Source code-only PRs tend to be specific and short

A small fraction of PRs (7.5%) only refer to SOURCE CODE (Table 4.4). These PRs tend to include atomic changes in the code with restricted impact like changing variable names, adding arguments, updating properties and so on. Therefore, these

PRs tend to reference many *Code Elements* in their comments and their discussion threads typically contain fewer than three comments.

4.2.2 Information Specificity

Apart from the type of information resource being referenced, the granularity or specificity of the information being referenced also influences the choice of references to be used during PR discussion. Based on the situation, developers tend to adjust the degree of specificity, for example, in detailed discussion related to code changes, they tend to discuss the specific line of code in a source code file, in other situations while discussing the overall impact on the class, they tend to refer to the source code file directly. For other types such as documentation and build releases developers tend to reference the specific part of the resource in their discussion. The PR discussion interfaces can be enhanced to support referencing and dereferencing these nested information types by changing the degree of specificity. Some of these practices which have been observed in our data are presented below.

Documentation references tend to include additional information along with URLs to denote specificity

DOCUMENTATION references, developers often point to a specific section of the referent. They tend to augment a written description to denote the exact location in their reference for readers to focus on. For example, a developer referring to a subsection of a document in a comment as "...the third section of README.md". Apart from referring to specific structural organisation of the document, developers also reference a specific content item in a project file such as " copyright year in License.md". These references are majorly expressed as plain text because nested sections of the information resource might not be located by a URL. Sometimes, developers combine plain text with links by augmenting additional textual information with document URLs. This saves the other discussants' time in locating the information and also prevents misunderstanding.

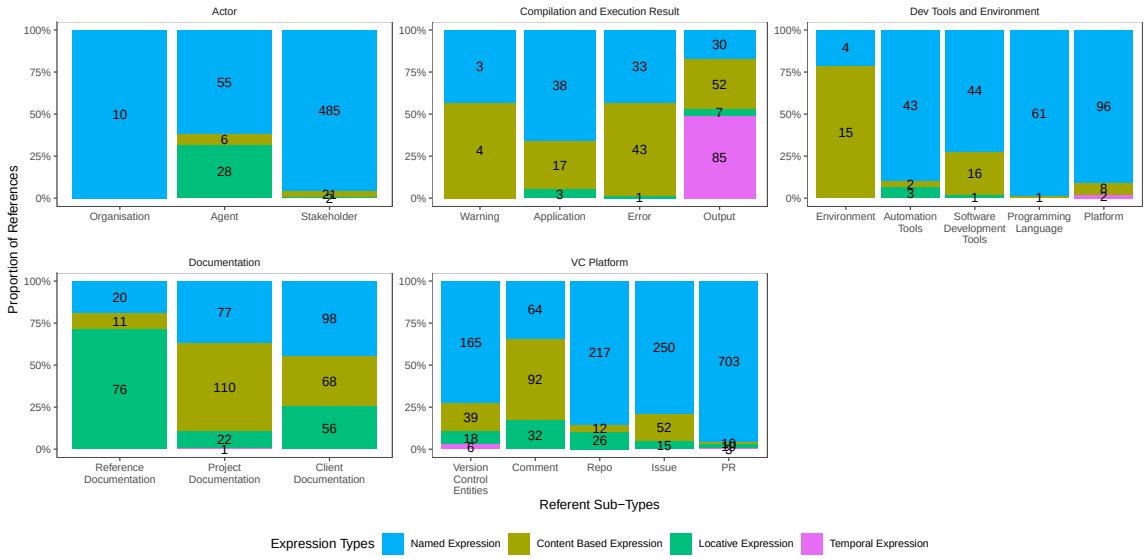

Figure 4.6: Distribution of expression types for non-source code referent types.

Developers tend to use temporal information for Build Release and Execution Output

Another pattern of using specific information for referencing is observed in references having temporal attributes like revision history. Many referent types such as COMPILATION AND EXECUTION RESULT and VC PLATFORM are expressed using their version number. For instance, one PR has a comment "The change is made in a way to support both Butterknife 7 and older versions", which refers to the release version of the software. This temporal information is useful to refer to a specific release of the software which is useful in evaluating the impact of changes submitted in the PR on backward and forward compatibility with the software. 85 out of 174 (48.9%) references to *Output* artifacts like build releases explicitly employ temporal expressions to increase their degree of specificity (see Figure 4.6). Knowing temporal information of these referent types is necessary for reviewers to gauge the impact on the code base and helps in prioritizing PRs.

4.2.3 Cultural Practices

The software engineering culture and practices influence the referencing patterns in the PR discussion. For example, the project type, programming language used in the project impacts the software development practices of the project such as development, testing, reporting issues. It ultimately influences the discussion and the references used in the discussion as well. These cultural factors can inform the design of PR discussion interfaces to be customizable to allow referencing and dereferencing resources based on the type of the project under discussion.

Developers tend to refer to internal documentation by name and external documentation by URL

76 out of 107 (71%) references to the *Reference Documentation* which are external to the project environment (e.g., third-party web pages, forum threads, specification websites) are generally referenced by locative expressions (i.e. using URLs) (see Figure 4.6). These information resources exist outside the shared context of the discussion, therefore, to ground them for every participant in the thread, a URL link to the referent is shared in the comments. On the other hand, discussants reference daily-use documents that are internal to the project such as *Project Documentation* and *Client Documentation* majorly by names. An explicit link to these resources is not provided as they are already part of the shared understanding.

UI references are common in projects which have user interface components

Projects such as Android, iOS, and web development frequently reference a lot of UI elements. 6.3% of references in these projects are related to UI referent types as compared to other projects which contain only 0.7% UI related references. We also found that these referents are expressed using descriptive names or short phrases. For example, in one PR discussion referencing a button in the GUI, participants referred to it by its screen name "show more button" Also, to express transient information (e.g., UI effects, states, and transitions) which cannot be pointed like a button in the above example, participants use GIFs or images in pairs to show before and after scenarios.

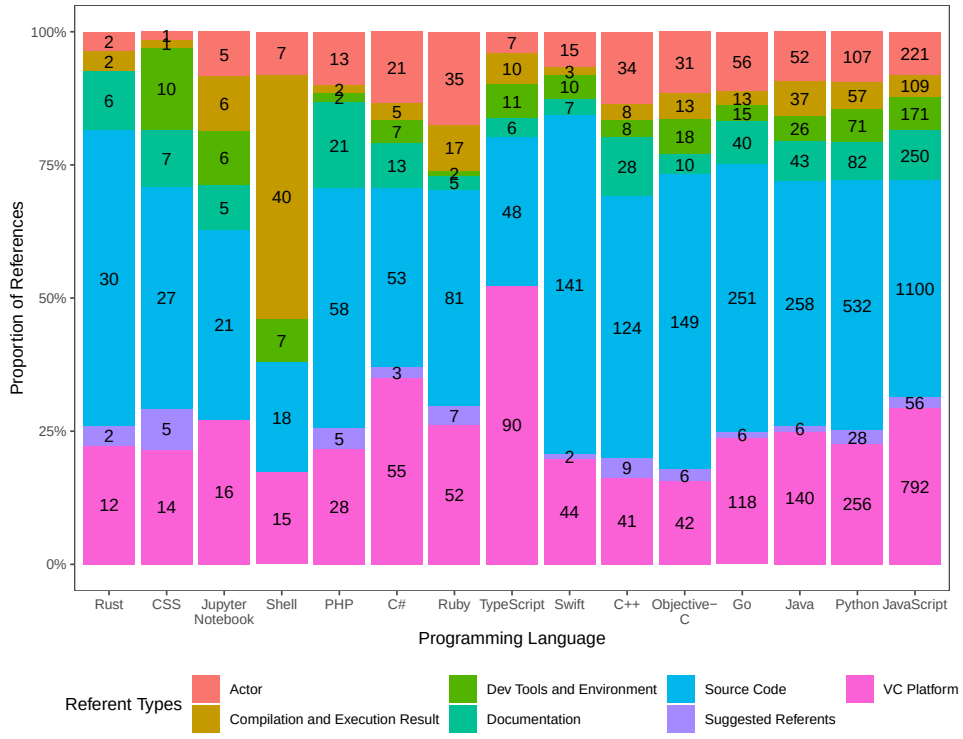

Figure 4.7: Distribution of referent types by project programming language.

Reference usage is correlated with the programming language of the project

We found that there is a significant relationship between a programming language and referent types, $\chi^2(70, n = 6558) = 648.25, p < .001$. A programming language is assigned to each repository by GitHub, which we collected during the sampling process. The amount of DEV TOOLS AND ENVIRONMENT references in PR discussions of projects using popular languages (e.g., JavaScript and Python) is more frequent as compared to less popular languages (e.g., Ruby and Rust) (see Figure 4.7). The popularity of a language is based on surveys conducted by GitHub[1] and Stackoverflow[2]. For example, PRs on JavaScript projects, reference build tools and IDEs. Various *Environments* are also referenced, like operating systems and web browsers to discuss compatibility issues.

[1] https://octoverse.github.com/

[2] https://insights.stackoverflow.com/survey/2020

Developers commonly use suggested references in projects of web-based programming languages

There is another dimension of referents that we identified in PR discussions, references to information resources that do not yet exist in the software. We call these *Suggested Referents*. Any referent type in our taxonomy Table 3.2 can be a suggested referent. We found that suggested referents are commonly used in projects with primary language as JavaScript, CSS, PHP and Python (see Figure 4.7). For example, in one PR in a CSS project, a reviewer commented "If you could update this to data-balloon-* attributes instead of classes, I would be glad to accept this PR". data-balloon-* is a suggested property in the comment. A web-based programming language such as CSS offers many ways to implement the same feature in a relatively few lines of code, we suspect that the expressive nature of CSS is responsible for the pattern we observed. The increase in suggested referents also indicates that developers tend to have longer discussions in such projects related to alternative solutions and suggestions. Also, the number of suggested referent types are found more in closed PRs as compared to merged PRs which indicates that the discussion has led to re-work in the PR submission.

4.2.4 Social Norms

PR discussion is a social activity which involves collaborative discussion carried out by users of the platform to reach to a conclusion. The references are influenced by the social norms of the population group which are carrying the discussion. For example, users are referenced in the discussion for a variety of reasons such as to question, to reply, to seek clarification, to add them into the discussion thread, or for informational purposes only. Similarly, the group of users carrying the discussion also use generic references to refer to resources in the discourse which is an acceptable social norm. These findings are discussed in detail below.

Developers frequently use generic terms to reference information scoped within the discussion thread

We found that 30.2% of the references in our dataset are expressed using *generic* terms such as pronouns, adjectives, and collective nouns (Figure 4.8). These refer-

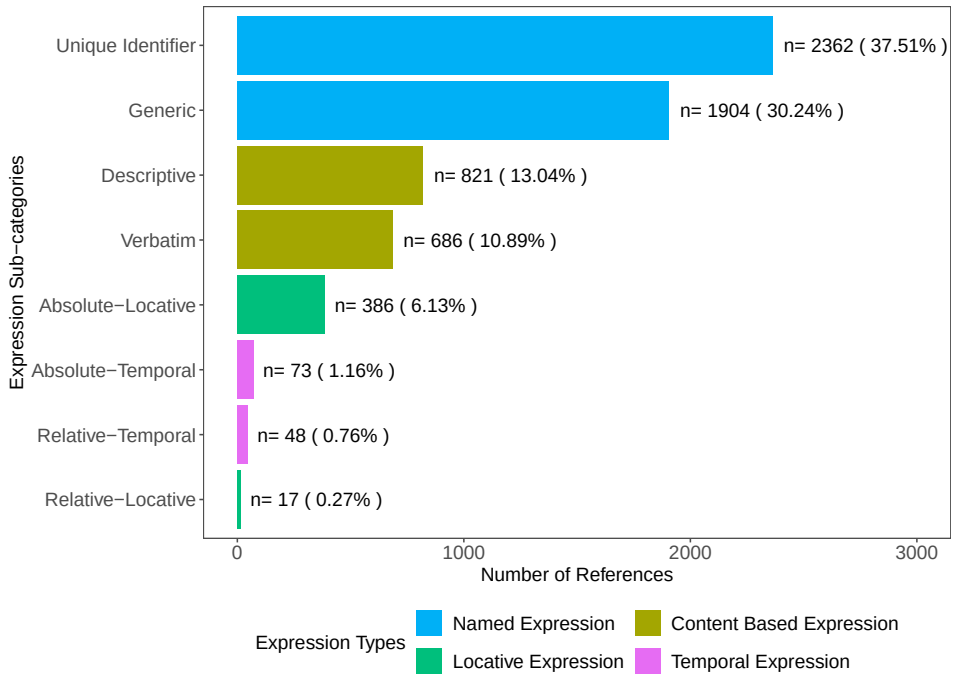

Figure 4.8: Distribution of expression sub-categories.

ences are used for various purposes. Once a reference is introduced in a comment, it is re-referenced in the discourse repeatedly using generic terms like pronouns (it, this, etc.). For example, after introducing a class by its name "EnhancedFactory-Data", it was expressed in the discourse using pronouns, such as "it" and "this". This works because the mutual understanding of the referential identity is already established, there is no need to reference it by its name again, therefore pronouns like "this" can be used. In other scenarios, when the referent is obvious in the discussion, such as the PR itself or the issue under discussion, generic terms like demonstrative adjectives (i.e., "This PR", "This issue") are used. For example, a submitter referenced the self PR in a comment as "This fixes #9". Also, when the discussant points to a collective group rather than to a specific referent, for example, referring to the "contributors" instead of a specific user, collective nouns are used as generic expressions. All these social norms of using language impacts how the references are made in the PR discussion. References with generic terms may

Table 4.5: Impact of number of comments and participants in @-mention references in a PR (Results of Negative Binomial Regression Test)

	Estimate	**Standard Error**	**p-Value**
Intercept	-2.09	.163	$<.001^{***}$
comments_count	.24	.017	$<.001^{***}$
participants_count	.07	.046	.108

work well for those who are following the discussion but may pose a problem for new readers, who need to re-read the discussion from the top to understand the generic reference. Also, having multiple referents expressed with generic words in the same discussion may lead to misunderstanding.

Stakeholder references (@-mentions) are prevalent in long PR discussions

References to ACTOR, especially *Stakeholders* (i.e., contributors, reviewers, participants), tend to increase with the number of the comments in the PR. To test this hypothesis, we used Negative Binomial Regression as it is suitable for over-dispersed count variables (see Table 4.5). We chose number of comments and number of participants in a PR as the predictor variables because these factors can increase the likelihood of referencing users (@-mention) in a discussion thread. Out of these two factors, we found that there is a significant relationship between number of comments and number of @-mentions in a PR discussion (see Table 4.5). With the increase in the number of comments, the number of @-mention references tend to increase in a thread. This result adds inferential evidence to a similar observation reported in the previous work [68].

By closely reading the PRs, we found that stakeholders are referenced for various purposes in comments using @-mention. The most common reason for referencing users is to discuss the content of the PR such as replying to other user's comments, directing questions to a user, appreciating a user's response or contribution. For example, in one PR comment, a reviewer is asking for the rationale of the new API as "Hi @georgiosd. Can you describe the use case/real-world scenario for "should retry" in more detail, so that I can better understand? Thanks!" Other times, we found that @-mentions are used for tagging users for administrative pur-

poses such as requesting a review, inviting a user to participate in the discussion, and tagging a user for information purposes only. For example, a user was tagged for notification purposes in the discussion by commenting "/cc @mabrahamde". Also, people express their agreement and disagreement on the suggested solutions during the discussion such as "I agree with @st0le's way as well". Since most of the purposes are covered by @-mention in comments, but there are some reasons for which developers do not use @-mentions. For example, to reference a user who is not on GitHub, the name or email address of the user is mentioned. And, to refer to the group of users collectively, such as all the contributors in the project, developers mention them in plain text as in this comment "This way contributors don't have to waste time in setting up the development environment and getting the dependencies installed."

4.3 Referencing patterns in PRs with different outcomes

At the end of the day, a successful PR is either merged or closed because reaching a conclusion quickly is important for efficiency in the software development process. Therefore, the essence of the discussion process in the PR is to evaluate the contribution and reach an outcome. The use of references in a PR discussion can help in advancing the discussion process so that the PR progresses towards conclusion. We start this section with an analysis of referencing patterns observed in merged and closed PRs, because these two statuses are conclusive while the open status is indefinite, where things are left to be decided. In the second subsection, we briefly share our high-level observations from open PRs.

4.3.1 Accepted or Rejected PRs tend to have references to the evidence that advocates their appropriateness.

References to *Tests*, *Agents*, and *Issues* are more frequently found in merged than closed PRs (see Figure 4.9). These referent types indicate that the participants in the discussion may have taken steps to ensure that the changes in a PR do not impact the project negatively or the changes are important for maintaining the quality of the main repository. For example, references to unit tests and test suites can signify the presence of tests and testing code, which make it easy to verify the cor-

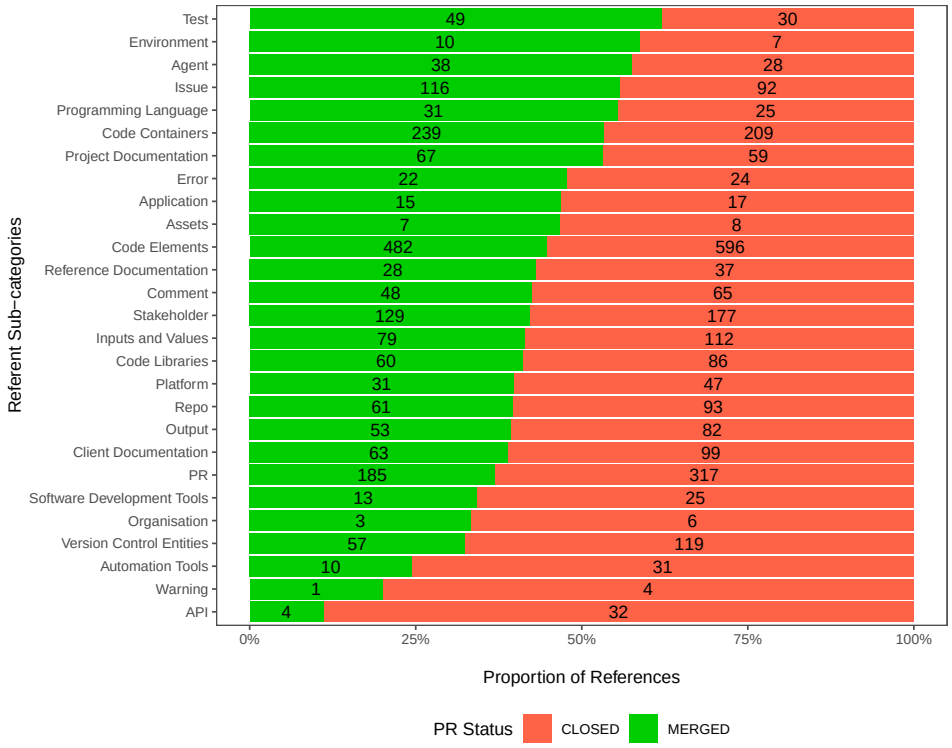

Figure 4.9: Distribution of all referent types by PR status (closed/merged).

rectness of the pull request. The presence of Agent references can indicate the use of a bot or third party service for running CI loops, calculating code coverage, and so on, which provides further evidence that the PR has been reviewed for quality. For example, a kubernetes-prow-bot has been used in the ”Kubernetes” project in our dataset; it can run test suites, add labels to the PR, and merge the code based on users’ commands. Similarly, the ”CLAassistant” bot is used to ensure contributors have signed the contributor license agreement. Although, *Environment* references are also higher in merged PRs than closed, we refrain from making strong claims from them due to the small sample size with only 17 instances.

References to *Automation Tools* and *Version Control Entities* are more than twice likely to be found in closed PRs than merged PRs (see Figure 4.9). These pattern indicates that the discussion may have taken place regarding the source code management aspect of software development such as using a wrong branch, merge

conflicts, build related issues, or redundant fixes. *Automation Tools* references in closed PR discussions may indicate the negative outcomes of a build system on a given PR [69]. Similarly, *Version Control Entities* references may appear to indicate duplicate fixes or pending issues in the submission which is preventing a proposed change from being merged. All these factors are indicative of how certain referent types are more frequently found in closed PRs than others. We also found that majority of API-heavy PRs end up closing (N = 32) than merging (N = 4) because such discussions tend to be contentious and could easily lead to rework in most cases (see Figure 4.9). However, we found that a similar number of API references (N = 33) are also present in open PRs which are yet to reach a conclusion which indicates that these PRs tend to remain inconclusive before closing. Therefore, we present our observations on open PRs separately in the next section.

4.3.2 In Open PRs, developers seem to be discussing high-level changes such as API and Compatibility issues in software projects

We found that relatively high proportion of API-level changes in the software or compatibility issues are found in open PRs (33 out of 69 API instances) than the other two types (4 in merged, 32 in closed). These PRs generally involve discussion about how the proposed changes may affect the entire project, for example, with regards to backward compatibility and support for running on different platforms. This is difficult to appraise. For example, a sampled PR[3] in the "gRPC" repository proposed an API for client interceptors in which discussion carried onto discussing its compatibility on different runtime platforms and API design in detail amongst the four discussants. The discussion ended without any decision around the time we sampled this PR, and 5 months later the PR was closed with a message that the proposed changes were in conflict with the long-term API goals of the project. Such PRs are more likely to remain in an open state, because consensus is difficult to reach.

[3] https://github.com/grpc/grpc-web/pull/558

Chapter 5

Learnings from Referencing

Our thesis provides insights into developer's referencing practices in PR discussions; however, consolidating our findings establishes two major takeaways for the design and research community. First, many prevalent referencing practices generate design implications for supporting rich referencing in PR discussion interfaces which are discussed in Section 5.1. Second, the study of referencing patterns in PRs opens up new possibilities for further research to explore its impact on the flow of discussion and ultimately on the discussion outcome which we elaborate in Section 5.2 and Section 5.3. Finally, we end the chapter with a discussion of how prevalent is the lack of referencing support in communication occurring beyond the context of PRs. Table 5.1 summarizes the key design implications and research opportunities derived from the findings in this thesis.

5.1 Design Implications

We identified various referencing patterns in PR discussions that are not supported by GitHub's auto-linked referencing interface. For example, source code which is a frequent object of discussion in PRs, is primarily discussed in plain text. Similarly, there is no support for referencing UI elements, nested content in the documentation, build releases, and certain resources that appear within the scope of the discussion thread itself. In this section, we discuss the implications for design generated through the referencing practices observed in PR discussions to better sup-

Table 5.1: Summary of learnings derived from studying referencing patterns in PRs.

	Design Implications
1	In-situ code editing for source code referencing.
2	Built-in live app-emulation for UI referencing in PRs.
3	Referencing nested content in docs using transclusion.
4	Auto-detection of references generated within PR thread.
5	Referencing build releases, and project docs with auto-linking.
	Research Opportunities
6	Exploring the relationship between referencing and PR outcomes.
7	Exploring the use of references to impact the stage of discussion.
8	Extending the support of referencing to discussions beyond software engineering domains.

port referencing and dereferencing various underrepresented referent types through GitHub's PR discussion interface.

5.1.1 There is a need for referencing finer details in source code

Our findings in Section 4.1.3 indicate that *Code Elements* in the SOURCE CODE have been the major part of a discussion in PRs. However, developers are making these references by typing text which is tedious and error-prone. Especially in review comments, typing correct variables names, or correct signature of the function is important because that is the only index to the actual code in the file. Although GitHub renders the code differently from the regular text in the comments which provides a visual distinction, it does not support developers in authoring code within comments. This suggests the need for in-situ code editing capabilities within the PR interface which can assist developers to reference and dereference various code elements in the source code within the repository. Like a code editor, the interface will provide content assist (aka code completion) to write code faster and efficiently. When a developer types in the first few letters, it provides a list of matching code elements (variables, functions, classes, etc.) pulled from the source code within the project environment. The developer can make a selection from the list and the interface will automatically add a link to the referent in the source code file in the repository. It allows developers to type code with few keystrokes as

well as maintain accuracy. It is even more convenient when functions and variables with the same name exist in different code files as it links to the right definition. These auto-linked code references can be dereferenced in many ways. By hovering over the reference (e.g. a function name) in the comment, the interface displays its function definition in a tooltip right within the PR thread. Or by clicking the reference, one can navigate to its location in the code file. The in-situ code-editing capabilities within the commenting interface of PR discussion can speed up coding authoring within comments and improves efficiency by reducing typos and other common mistakes.

5.1.2 Referencing UI elements in PR Discussion

We found that projects with interface elements reference various resources such as UI elements, layouts, colors, positioning in their text comments. These resources require long text to describe as well as their description style varies from one person to another. It becomes even more difficult to reference information like state changes, animations, background effects that cannot be pointed out on the interface. Developers interleave screenshot images with a text description in their comments which is unnecessarily time-consuming. To reproduce issues, or demonstrate the implemented changes, developers reference links to the running instance of the app on cloud IDEs such as StackBlitz[1], Gitpod[2]. The reviewer can browse these hosted apps to interact with the live application. However, these are situated outside the PR's discussion environment and require context switching. This suggests there is a need for built-in UI emulation with referencing and dereferencing capabilities within the PR discussion interface. The emulator runs the app next to the discussion thread in the PR and allows users to access different UI elements in the app. For example, to reference a button in the app, the user can select the button on the UI and drag it into the commenting interface which will automatically add a link to it in the text comment. Developers can also take the screenshot of the UI right within the PR interface which prevents them from context switching. These linked UI references in the comments can be dereferenced by clicking the link which loads the UI in the emulator next to it.

[1] https://stackblitz.com/

[2] https://gitpod.io/workspaces/

5.1.3 Nested referencing in comments can be supported with transclusion.

We found that references were often made to specific elements of information objects. For example, references to documentation, code files, web pages, often include a location within the larger information object. For example, "The third section of readme.md" specifies a section in a document. In previous work on hypertext, there have been calls for transclusions [38, 51]. Transclusion is a method of including all or part of an information object into another, which eliminates the need to dereference. However, more recent work has highlighted challenges for readers making sense of these types of links because key context can be missing in transclusions [36, 44]. This implies that tools should support forms of transclusion appropriate to the granularity of the content with particular contexts, such as text sections and code snippets, found in PR discussions. Chua et al. [13] suggested one of few solutions to the nested referencing for varying degrees of specificity, addressing the specific case for documents in the online learning context.

5.1.4 There is a need for supporting references within a PR discussion thread

We found that various references are being generated within the scope of the discussion thread itself. For example, developers introduce issues, APIs, and features with short titles like "typescript bug", "flake8 issue", "Timing API", etc. These references once introduced in the scope of discussion are continued to be used by other people in their replies. However, there is no mechanism to trace the location where these are initially created. Therefore, users who are not actively involved in the discussion need to read the complete thread from the beginning to trace the definition of these references. These references become even more difficult to decipher when one comment of the PR is cross-referenced into another PR. This suggests a need for an auto-detection feature within the PR discussion interface that can automatically highlight these references in the discussion thread and link the comment in which it was introduced. The user can navigate directly to the comment where the reference was introduced by clicking the reference without needing to read the whole thread.

5.1.5 Extending GitHub's current support for auto-linking software deliverables

We noted in Section 4.2.2 that developers make text references to software deliverables such as build releases, project documents in their comments during discussion. Although, there has been a way to link URLs to these resources by copy-pasting it in the text, but it has not been used effectively because it is time-consuming and requires context switching. Since GitHub's auto-linking feature already supports referencing many platform-specific entities such as Issues, PRs, and users. It can be easily extended to support other resources that are intrinsic to projects such as build releases, project files, etc. To reference a build release, a user can type a prefix character which invokes quick navigation within the authoring interface. The quick navigation provides the list of releases published by this repository on GitHub. The user can navigate the list and make a selection and a linked reference to the release is added in the text comments. This feature enhances a common linking feature provided by GitHub to autolink any external resources[3]. The reference includes an embedded link to the release page which can be dereferenced by the readers from the discussion interface directly.

5.2 Referencing and discussion stages

During a code review process, the PR discussion goes through different discussion stages. Referencing can act as a benchmark to the stage of discussion. For example, the title and description provided by the submitter in the PR sets the grounds for reviewers to start the discussion. Based on the references given in the description, a reviewer can decide the next course of action to either continue with the code review or ask for additional information from the submitter. Similarly, in case of disagreement with the submitted change, a reviewer may suggest an alternative change to the code, or decide to reject it. In these scenarios, a discussion moves into different stages and the people in the discussion make different references in the comments over time. These references in the discussion can orient the flow of the PR discussion in a specific direction. Therefore, it is interesting to explore what set of references can impact the PR discussion to progress in a

[3] https://docs.github.com/en/github/administering-a-repository/configuring-autolinks-to-reference-external-resources

forwarding direction. This kind of study can also reveal many commonly occurring behaviors in these conversations, as well as well-established norms that can have many utilitarian implications. For example, what kind of references people make in their comments to come to an agreement can inform the design of an automated bot which can facilitate the decision-making process by making similar comments. Also, knowing the stage of discussion, we can build predictive models to predict when a PR can reach a conclusion which will be effective in managing the efficiency of software development process.

5.3 Referencing and PR outcomes

Our findings indicate various referencing patterns found in PRs with different outcomes. This observation suggests that analysing referencing is an important yet understudied aspect of code review discussion. Discussions that use references in different ways may lead to different, correlated outcomes, depending on the context. For example, Yarmand et al. [64] showed that referencing timestamped video snippets in comments can increase the number of "likes" and replies on YouTube. Similarly, Liu et al. [40] found that messages that reference multimedia content on Twitter receive more re-tweets. On StackOverflow, answers with references receive more "up" votes [50]. Note that having certain references in a PR may not at all ensure that it will be accepted because many technical and social factors impact the PR outcome [26, 27, 59, 60, 65]. References can reduce the cognitive workload in a discussion which helps to reach a conclusion faster. Therefore, it is interesting to see in future work how references can influence a certain outcome over time. This can be done by building a sophisticated statistical model that can predict the PR outcome based on various factors including the references. Together, the findings across this and previous work hint at the importance of studying and improving the referencing and dereferencing capabilities of our software to better support collaboration and discussion.

5.4 Varying levels of platform's support for referencing

GitHub is an evolving platform. Over time, new referencing features have been added to its interface. For example, in 2011, GitHub introduced auto-linking users

using @-mention in their PR interface [3]. Similarly, in 2019, GitHub introduced a custom prefix feature that enables the users to enable a hyperlink to new referent types, but the feature is available for the enterprise tier users only [57]. In Chapter 4, we only captured referencing behaviors that took place when the users were given only a type of referencing feature set. One of the potential future work is to study how different levels of platform's interface support may impact the way people refers to information resources in their discussions by comparing people's referencing behaviors before and after GitHub's auto-linking feature was introduced. Another study can focus on practices for creating and using different custom prefixes. Therefore, referencing feature set could become an another sampling criteria in data collection for future studies.

5.5 General lack of referencing support

While we have established that referencing is an important tool for effective communication and that rich referencing is occurring in PR discussions, we also found that support for it is limited. Fewer than 16% of the references in our dataset are supported by the GitHub platform. Despite lacking support, developers are introducing textual references to e.g., a shared codebase, other repositories on GitHub, various third-party websites, suggesting the importance of references in establishing common ground. Since GitHub's referencing interface already supports VC PLATFORM, therefore, support for other platform-entities like repositories can be easily extended. Similarly, other studies have also found that general support for referencing is lacking in contexts beyond code review. Yarmand et al. [64], for example, found that most referencing behaviors are unsupported on YouTube. We suspect that this is a concern for other Q&A platforms, such as StackOverflow and Quora, where users lack the same degree of shared context as on GitHub. In these contexts referencing support could be crucial for effective communication [50].

5.6 Summary

In summary, the study of referencing patterns in PR discussions have many implications for the design and the research community in general. We discussed design ideas to support referencing in PR discussion interfaces which facilitate effective

communication during the code review process. We also discussed how the study of referencing has promising potential to explore as another influencing factor for PR outcomes. We believe the work of understanding PR referencing behavior has just begun. Future work, for example, could analyze how the use of references maneuver the flow of discussion which can generate more design guidelines for the community. In the end, we highlighted the general scarcity in supporting referencing in discussions beyond PR settings as well.

Chapter 6

Conclusion

In this thesis we examined software developers' referential behaviors in code reviews through an analysis of ∼7K annotated references from 450 public PRs sampled from GitHub. We provided an analysis of the patterns of what people refer to (type) and how they refer it (expression). We provided the detailed empirical accounts of how well the referencing is supported by GitHub. We found source code is the most frequently referenced type of information, even though it is under-supported by GitHub's referencing interface. We found information type, information specificity, cultural practices, and social norms are the four contextual factors that shape the use of references in PR discussions. We presented several referencing practices shaped by these factors. Finally, we found distinct referential behaviors in merged, closed and open PRs. Based on consolidating our analyses, we provided design implications to support effective referencing in PR discussion interfaces. We also discussed some more general insights about referencing behaviour in PR discussions and reflected on how it promises potential for further research.

This study has several limitations. With regards to the taxonomy, we focused only on referent types and referential expressions. Other important aspects of the referencing behaviors fall outside the scope of the current study. We collected a diversified sample of PR discussions from popular repositories on GitHub, based on star ratings. Future work could prioritize other repository attributes, such as number of contributors. Our dataset does not contain PR discussions from private

repositories on GitHub, leaving questions about the referencing practices within closed-source projects from our analysis.

This study created a data-driven taxonomy of references stakeholders make in PR discussions, analyzed the existing referencing support provided by GitHub and observed several patterns of referential behaviors in practice. Through the articulation of the PR reference behavior in this paper, better software tools can be designed to accommodate the rich nature of referencing in stakeholder PR discussions to improve source code design, implementation, management and, ultimately, trustworthiness.

Bibliography

[1] Features. Code Review. GitHub, 2019. URL https://github.com/features/code-review/. Accessed May 5, 2019. → pages 1, 10

[2] The state of the Octoverse: An overview on community and platform growth, 2019. URL https://octoverse.github.com/2019/. Accessed Nov. 6, 2019. → page 1

[3] Autolinked references and URLs - GitHub Docs, 2020. URL https://docs.github.com/en/github/writing-on-github/autolinked-references-and-urls. Accessed Oct 29, 2020. → pages 3, 22, 50

[4] GitHub: Where the world builds software, 2021. URL https://github.com/. Accessed Feb. 15, 2021. → page 10

[5] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721. IEEE, 2013. → page 2

[6] L. Baron, J. Tague-Sutcliffe, M. T. Kinnucan, and T. Carey. Labeled, typed links as cues when reading hypertext documents. *Journal of the American Society for Information Science*, 47(12):896–908, 1996. → page 9

[7] A. Begel. Program commenting by voice, 2002. → page 11

[8] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens. Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 202–211. Association for Computing Machinery, 2014. ISBN 9781450328630. → page 9

[9] H. Borges and M. T. Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018. → page 14

[10] H. Borges, A. Hora, and M. T. Valente. Predicting the popularity of github repositories. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 1–10, 2016. → page 14

[11] Y.-C. Chang, H.-C. Wang, H.-k. Chu, S.-Y. Lin, and S.-P. Wang. Alpharead: Support unambiguous referencing in remote collaboration with readable object annotation. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 2246–2259, 2017. → page 7

[12] Y. Chen, S. W. Lee, Y. Xie, Y. Yang, W. S. Lasecki, and S. Oney. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6220–6231, 2017. → page 11

[13] S. H. Chua, T.-J. K. P. Monserrat, D. Yoon, J. Kim, and S. Zhao. Korero: Facilitating complex referencing of visual materials in asynchronous discussion interface. *Proceedings of the ACM on Human-Computer Interaction*, 1(CSCW):1–19, 2017. → pages 3, 4, 7, 47

[14] H. Clark. *Using Language*. Cambridge University Press, Cambridge, United Kingdom, 1996. → pages 2, 7

[15] H. Clark and S. E. Brennan. Grounding in communication. 1991. → pages 2, 6, 8

[16] H. Clark, R. Schreuder, and S. Buttrick. Common ground at the understanding of demonstrative reference. *Journal of Verbal Learning and Verbal Behavior*, 22(2):245–258, 1983. → page 7

[17] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. → page 18

[18] V. Cosentino, J. L. C. Izquierdo, and J. Cabot. Findings from github: methods, datasets and limitations. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 137–141. IEEE, 2016. → page 2

[19] C. Courtright. Context in information behavior research. *Annual review of information science and technology*, 41(1):273–306, 2007. → page 30

[20] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. Communicative intention in code review questions. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 519–523. IEEE, 2018. → page 8

[21] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 49–60, 2019. → page 9

[22] V. Efstathiou and D. Spinellis. Code review comments: language matters. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 69–72, 2018. → page 7

[23] S. R. Fussell, R. E. Kraut, and J. Siegel. Coordination of communication: Effects of shared visual context on collaborative work. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 21–30, 2000. → pages 3, 4

[24] G. Gousios. The ghtorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236. IEEE, 2013. → page 10

[25] G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371, 2014. → page 10

[26] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 345–355, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327565. → pages 1, 2, 3, 8, 20, 49

[27] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, page 358–368. IEEE Press, 2015. ISBN 9781479919345. → pages 3, 8, 10, 20, 49

[28] D. Greyson, H. O'Brien, and S. Shankar. Visual analysis of information world maps: An exploration of four methods. *Journal of Information Science*, 46(3):361–377, 2020. → page 13

[29] Y. Hao, G. Li, L. Mou, L. Zhang, and Z. Jin. Mct: A tool for commenting programs by multimedia comments. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1339–1342. IEEE, 2013. → page 11

[30] A. Z. Henley, K. Muçlu, M. Christakis, S. D. Fleming, and C. Bird. Cfar: A tool to increase communication, productivity, and review quality in collaborative code reviews. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018. → page 11

[31] F. Husson, J. Josse, and J. Pagès. Principal component methods-hierarchical clustering-partitional clustering: Why would we need to choose for visualizing data?, 2010. → page 21

[32] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101, 2014. → page 10

[33] O. Kononenko, O. Baysal, and M. W. Godfrey. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1028–1038, 2016. → pages 7, 8

[34] R. W. Kopak. Functional link typing in hypertext. *ACM Computing Surveys (CSUR)*, 31(4es):16–es, 1999. → page 9

[35] K. Krippendorff. *Content Analysis: An Introduction to Its Methodology*. SAGE, Los Angeles, CA, 2019. → page 13

[36] H. Krottmaier and H. A. Maurer. Transclusions in the 21st century. *Journal of Universal Computer Science*, 7(12):1125–1136, 2001. → page 47

[37] A. Kunert, A. Kulik, S. Beck, and B. Froehlich. Photoportals: shared references in space and time. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 1388–1399, 2014. → page 7

[38] G. Landow. *Hypertext 3.0: Critical Theory and New Media in an Era of Globalization*. Johns Hopkins University Press, Baltimore, MD, 2006. → page 47

[39] Z.-X. Li, Y. Yu, G. Yin, T. Wang, and H.-M. Wang. What are they talking about? analyzing code reviews in pull-based development model. *Journal of Computer Science and Technology*, 32(6):1060–1075, 2017. → page 9

[40] Z. Liu, L. Liu, and H. Li. Determinants of information retweeting in microblogging. *Internet Research*, 2012. → page 49

[41] A. Madden, I. Ruthven, and D. McMenemy. A classification scheme for content analyses of youtube video comments. *Journal of Documentation*, 69 (5):693–714, 2013. → page 13

[42] E. E. Marsh and M. D. White. A taxonomy of relationships between images and text. *Journal of Documentation*, 59(6):647–672, 2003. → pages 9, 13

[43] C. Marshall. Toward an ecology of hypertext annotation. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, pages 40–49, New York, NY, 1998. ACM. → page 4

[44] H. Maurer and J. Kolbitsch. Transclusions in an html-based environment. *Journal of Computing and Information Technology*, 14(2):161–173, 2006. → page 47

[45] N. McDonald, S. Schoenebeck, and A. Forte. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for cscw and hci practice. *Proceedings of the ACM on Human-Computer Interaction*, 3 (CSCW):1–23, 2019. → page 18

[46] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21(5):2146–2189, 2016. → page 7

[47] M. Mühlpfordt and M. Wessner. Explicit referencing in chat supports collaborative learning. 2005. → page 7

[48] M. V. Mäntylä and C. Lassenius. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3): 430–448, 2009. → page 9

[49] M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 466–476, New York, NY, 2013. ACM. → page 16

[50] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012. → pages 49, 50

[51] T. Nelson. *Computer Lib/dream Machine*. Tempus Books of Microsoft Press, Redmond, WA, 1987. → page 47

[52] R. Padhye, S. Mani, and V. S. Sinha. A study of external community contribution to open-source projects on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 332–335, 2014. → page 14

[53] S. Park, A. X. Zhang, and D. R. Karger. Post-literate programming: Linking discussion and code in software development teams. In *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*, pages 51–53, 2018. → page 11

[54] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli. Information needs in contemporary code review. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), Nov. 2018. → pages 2, 9

[55] M. M. Rahman and C. K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 364–367, 2014. → page 10

[56] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165, 2014. → page 14

[57] L. Schneider. Save time linking resources with autolink references, 2019. URL https://github.blog/2019-10-14-introducing-autolink-references/. Accessed Oct 29, 2020. → page 50

[58] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319, 2017. → page 14

[59] D. M. Soares, M. L. de Lima Júnior, L. Murta, and A. Plastino. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, page 1541–1546, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331968. → pages 3, 8, 20, 49

[60] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th*

International Conference on Software Engineering, ICSE 2014, page 356–366, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327565. → pages 3, 8, 20, 49

[61] L. R. Tucker. Intra-individual and inter-individual multidimensionality. In S. Messick and H. Gulliksen, editors, *Psychological Scaling: Theory and Applications*, pages 155–167. Wiley, New York, NY, 1960. → page 21

[62] J. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA, 1977. → page 21

[63] G. Viviani, C. Janik-Jones, M. Famelis, and G. Murphy. The structure of software design discussions. In *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 104–107. IEEE, 2018. → pages 7, 8

[64] M. Yarmand, D. Yoon, S. Dodson, I. Roll, and S. S. Fels. "can you believe [1:21]?!": Content and time-based reference patterns in video comments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 489:1–489:12, New York, NY, 2019. ACM. → pages 3, 4, 7, 9, 49, 50

[65] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. Wait for it: Determinants of pull request evaluation latency on github. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 367–371, 2015. → pages 3, 8, 10, 20, 49

[66] F. Zampetti, L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, and M. Lanza. How developers document pull requests with external references. In *2017 IEEE/ACM 25th International Conference on Program Comprehension*, pages 23–33, Piscataway, NJ, 2017. IEEE. → page 3

[67] Y. Zhang and B. Wildemuth. *Qualitative analysis of content*, pages 318–330. Libraries Unlimited, Santa Barbara, CA, 2 edition, 2017. → page 13

[68] Y. Zhang, H. Wang, G. Yin, T. Wang, and Y. Yu. Exploring the use of @-mention to assist software development in github. In *Proceedings of the 7th Asia-pacific Symposium on Internetware*, pages 83–92, 2015. → pages 3, 40

[69] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: A large-scale empirical study. In *IEEE/ACM International Conference on*

Automated Software Engineering, pages 60–71, Piscataway, NJ, 2017. IEEE. → page 43

[70] S. Zyto, D. Karger, M. Ackerman, and S. Mahajan. Successful classroom deployment of a social document annotation system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1883–1892, New York, NY, 2012. ACM. → page 3

Appendix A

References Dataset Schema

This chapter contains the schema for the annotated dataset of ∼7k references compiled and analyzed in this project. The dataset in CSV format is available in supplementary materials along with this thesis.

Table A.1: Dataset Schema

Column Name	Description
uid	A unique ID given to each reference row.
repo_name	Name and owner information of the repo on GitHub from which PR is sampled for the reference.
repo_primary_language	The primary language of the repository on GitHub.
repo_pr_count	Total number of PRs in the repository at the time of sampling.
repo_open_pr_count	Number of open PRs in the repository.
repo_closed_pr_count	Number of closed PRs in the repository.
repo_merged_pr_count	Number of merged PRs in the repository.
repo_contributor_count	Total number of contributors in the repo.
repo_star_count	Number of stars given to the repo.
uid	A unique ID given to each reference row.

Column Name	Description
repo_name	Name and owner information of the repo on GitHub from which PR is sampled for the reference.
repo_issue_count	Number of issues raised in the repo.
repo_fork_count	Number of forks of the repo.
repo_watchers_count	Number of watchers subscribed to the repo.
pr_number	The PR ID assigned to the PR from which the reference is identified.
pr_url	The URL of the PR.
pr_comments_count	Total number of comments in the PR including the first comment by the contributor.
pr_participants_count	Total number of participants in the PR discussion as provided by GitHub API.
pr_created_at	The date and time in ISO format at which the PR was created.
pr_ended_at	The date and time in ISO format at which the PR was merged or closed. Blank for open PRs.
pr_status	Status assigned to the PR (OPEN, MERGED, CLOSED) at the time of sampling.
reference	The reference (word or phrase) coded in the PR.
ref_code	The referent code assigned to the reference. All referent codes are available in codebook.
ref_sub_category	The referent sub-category assigned to the reference.
ref_category	The top-level referent category assigned to the reference.
exp_code	The expression code assigned to the reference. All expression codes are available in codebook.
exp_sub_category	The expression sub-category assigned to the reference.
exp_category	The top-level expression category assigned to the reference.

Column Name	Description
comment	The annotated comment from which reference is extracted. The location of reference is marked with <<>>
domain_topics	The comma separated list of domain keywords assigned to the repository based on the topic modeling and manual reading of the repository.

Appendix B

Codebook for Referent Types

This chapter is the collection of 196 codes generated for referent types identified in the PR discussions. These codes are organised by their referent type and sub-category.

Table B.1: Referent Codes arranged by Referent Type (Sub-category) from Taxonomy Table 3.2

Codes	*Description*
Source Code (Code Elements)	
REF-Arguments	Arguments provided to the function in source code.
REF-Artifact	A machine learning construct.
REF-Attribute	The properties specified in HTML tags
REF-Class	Class construct of source code
REF-Class	Constant Variable The constant variable defined inside class
REF-Class Name	The name of the class not the class body.
REF-Class Object	The instance object of the class.
REF-Code Annotations	Annotations used in the source code with @-
REF-Code Comment	Comments in the source code file.
REF-Code Style of Submitted Fix in Self PR	The style of code of submitted code in commits in PR.

REF-Coding Style	The style of the code written in the source code files.
REF-Component	Web equivalent of a class.
REF-CSS Backgorund Effect	Transition property to modify background effects during animation.
REF-CSS Class Name	Class Name in CSS language.
REF-CSS Property	Properties that are being referred in discussion
REF-CSS Selector	CSS Selectors like id based, class based etc.
REF-Data Structure	Any data structure under discussion like lists, maps, arrays etc.
REF-Data Type	Type of data like integer, string etc.
REF-Function	A function in the source code.
REF-Function Name	The name of the function in the source code file.
REF-Function Parameter	The parameters of the function in the source code file.
REF-Function Signature	The signature of the function with name, parameter and return type in the source code file.
REF-HTML Element	The HTML tags
REF-Interface	The interface construct in the source code.
REF-Keyboard Event	The event fired when keyboard keys are interacted with.
REF-Keywords	Any reserved word which is part of the progrramming language used in the source code.
REF-Package	The folder in which classes are grouped in source code.
REF-Property Name	The name of the property(variable).
REF-Struct	A programming language construct.
REF-Submitted Fix in Self PR	The change submitted in the PR in terms of Code.
REF-Symbols	The symbol used in the programming language like []*/ etc.
REF-UI Element	A user interface component like button, drop down etc.

REF-UI Element Property	The property of UI element like color, event, name etc.
REF-UI Event	An event raised when user interacts with UI components in the running application.
REF-URLs	Any URL mentioned in the discussion which is not classified in any other category.
REF-Variable	A variable in the source code file.
REF-HTTP Protocol Terms	The network protocol related references.
Source Code (Code Libraries)	
REF-Data Visualization library	External library for data visualization.
REF-Dependent Library	External libraries used in the source code.
REF-Framework Library	The external library used as the framework within the source code like Angular, react etc.
REF-Library Version	The version of the code library under discussion.
REF-Third Party Library	Any third party library used with the software project.
REF-Third Party testing library	A testing library used within the software
Source Code (Code Containers)	
REF-Build Script	The code used to build and compile the source code.
REF-ChangeLog Document	Document containing change history of the software.
REF-Code Fragment in File	A piece of code from source code file comprises of more than 1 LOC.
REF-Contribution Guideline File	Contribution guideline file in the repo.
REF-File	Any file resource in the source code repository.
REF-File Extension	The extension of the files.
REF-File Location	The location and specific path of the file being referred as referent.
REF-Files	The collection of files
REF-Folder	The folder containing the files.

REF-Fragment in Build Script	The codesnippet from the build script file.
REF-Fragment in Doc File	The snippet from the documentation file.
REF-Fragment of Build File	The code snippet from the build file.
REF-Line of Code	A single line of code from the source code file.
REF-Literal Value	A value given to any variable, property etc.
REF-Machine Learning Models	A model designed and trained in Machine learning algorithms.
REF-Multiple Lines of Code	More than one line of code from source code file.
REF-Script	A source code written with scripting language.
Source Code (Inputs and Values)	
REF-CMD Command	The commands used in command line.
REF-Color	Color specified in source code under discussion.
REF-Configurable Options	Properties or variables used to configure the system or software.
REF-Console Command Tags	Additional Parameters provided to console commands.
REF-Cookie	Additional file content saved in web browser.
REF-password	A string of characters given to access a resource
REF-Property Value	The value of the property.
REF-UI Element State	The state of UI like hidden, visible etc.
REF-Variable Value	The value given to the variable in the source code file.
Source Code (Test)	
REF-Test Suite	A collection of test cases.
REF-Benchmark Test Case	A kind of test case
REF-Test Case	A test case code file
REF-Unit Tests	A test that checks the low-level functionality of a source code, for example, testing the function.
Source Code (Asset)	
REF-Attachments	A resource uploaded in the PR comment.
REF-Icons	The assets used in the project.
REF-Image	The snapshot of the referent in graphical format.

REF-Software License Keys	The license key used by users to activate the features of the software to use.
Source Code (API)	
REF-API	API used in the code
Actor (Stakeholder	
REF-Application User	The user who use the application under development in the PR.
REF-Committers in Self PR	Committers in the PR who submitted the commits, mostly the author of PR.
REF-GitHub Account	The user account on GitHub.
REF-Twitter Handle	Twitter user ID
REF-User	The user on the GitHub platform mentioned in the discussion.
REF-User Email	The email address of the user mentioned in the discussion.
Actor (Agent)	
REF-Bot	A automated program act as a user in the PR discussion.
REF-Bot Command	Command used to invoke bot actions.
REF-Third Party CI Service	Online CI service used to run DevOps of the software development.
REF-Third party Services	Any online services used like CI, code coverage, bots etc.
Actor (Organisation)	
REF-Company	Any organisation mentioned in the comment like Google, Apple etc.
REF-Company which owns the repo	The organization that fathers the repo on GitHub.
REF-Review Committee of a company	The committee setup to review the code in PR.
VC Platform (Version Control Entities)	
REF-Code Version	The version of the code under discussion.

REF-Commit Author	Author of commit.
REF-Commit Comment	A message given to the commit.
REF-Commit in Other Repo	A commit from different repo in GitHub.
REF-Commit in Self PR	A commit from the same PR under discussion
REF-Commit in Self Repo	Commit in the repo in which PR is raised.
REF-Git Branch	The branch of Git under discussion
REF-Git Hooks	Hooks being implemented in Git configuration for the repo.
REF-Git Merge Conflicts	The code conflict arise in the Git during merging process.
REF-Git Tag Message	The description given during the time of tagging in Git.
REF-GitHub UI Compare Diff Screen	The GitHub UI screen where code diff is shown.
VC Platform (Issue)	
REF-Issue	An issue raised on GitHub.
REF-Issue in other Repo	An Issue from another repo on GitHub.
VC Platform (PR)	
REF-GitHub PR	A PR feature on GitHub.
REF-Github PR Label	The label assigned to the PR on GitHub.
REF-Other PR by same contributor	PR on GitHub raised by same contributor of the current PR.
REF-Other PR in Other Repo	A PR raised in another repository on GitHub.
REF-Other PR in same repo	A PR raised in same repository on GitHub.
REF-PR in Other Repo	A PR raised in another repository on GitHub.
REF-Self PR	The current PR under discussion.
VC Platform (Repo)	
REF-Forked Repo	The forked version of a repo on GitHub.
REF-Other Repo	An another repository on GitHub.
REF-Self Repo	The repo in which the current PR which is under discussion resides.

VC Platform (Comment)	
REF-An External comment quoted in current comment	A comment from other PR or Issue on GitHub platform.
REF-Comment in Issue	Comments in the Issue in GitHub.
REF-Comment in other PR thread	Comments in other PRs in GitHub.
REF-Comment in Other Repo's issue	Comment in issue of another repo on GitHub.
REF-Email Notification Quoted in the same comment	Email notification message referred in the comment.
REF-Footnote	Referring to something which is already mentioned in the comment and have additional info given in footnote.
REF-Phrase or Term used in previous comment	A phrase used in previous comment in PR thread.
REF-Phrase or term used in same comment	A phrase used in the same comment in the PR thread.
REF-Phrase used in Title	A phrase used in the title of the PR.
REF-Previous Comment in Same Thread	A comment in the same thread.
REF-Previous comment quoted in the same comment	A previous comment copied in the current comment while answering.
REF-Reference used in previous comment	Any reference used in previous comment in the same PR thread.
Dev Tools and Environment (Software Development Tools)	
REF-Dev Tools	Tools that are used by developers for software development like console tools, inspection tools.
REF-IDE	The software development tool for editing the source code.
REF-Online IDE	An online tool used to edit source code
REF-Plugin	An additional component added to the tools to enhance its features.

Dev Tools and Environment (Automation Tools)	
REF-Build Tools	Tools and libraries like compiler etc, used to build source code.
REF-Compiler	The tool used to compile source code into executable.
REF-Dev Dependency Library	External libraries which are not part of the code, but used for managing Software development
REF-Linter	A library used to check the code styling issues.
REF-Source Code Platform	Version control systems like GitHub, SourceForge, BitBucket etc.
REF-Third Party package management	A tool used to manage the dependent libaries like NPM etc.
REF-Third Party Version Control Software	VCS software like Git, CVS, Mercurial etc.
Dev Tools and Environment (Environment)	
REF-Command Line Platform	Command line platform like CMD, bash, shell etc.
REF-Emulator	A software program used to run the code to see the output, like web server.
REF-Environment Configuration	Configuration related to libraries and settings needed to setup the execution platform.
REF-Virtual OS Software	An OS like Linux etc installed on Virtual Machines.
Dev Tools and Environment (Platform)	
REF-Hardware	The hardware on which the code will be executed.
REF-Mobile OS Platform	Mobile OS like Android, iOS under discussion.
REF-OS Platform	Any System OS like Windows, Mac etc.
REF-Runtime Platform	The platform on which the source code is executed, for example, OS, web server etc.
REF-Software Platform	Software based platform like Java Runtime, .NET CLI etc.

REF-Web Browser	A software used to access web pages on Internet.
Dev Tools and Environment (Programming Language)	
REF-Markup Language	The declarative language like HTML, XML.
REF-Programming Language	The language use to code the software.
Compilation and Execution Result (Error)	
REF-Compilation Error	The error received during the compilation of source code.
REF-Network Error	Error received on Network.
REF-Runtime Error	An error occurred during the execution of the source code.
REF-Test Case Execution Error	The error received after running the test case.
REF-UI Error	The error received in UI code execution.
REF-Unit Test Error	Error received while running the unit tests.
Compilation and Execution Result (Warning)	
REF-Compilation Warning	The warning message received during the compilation of source code.
Compilation and Execution Result (Output)	
REF-Build	Output given by compiling the code.
REF-Console Output	The output of the console commands.
REF-Execution Output	The output of any execution of the code like compilation, execution, function execution etc.
REF-Part of Execution Output	A snippet from execution output, for example, part of compilation error
REF-Release Version	The version of the release build of the software under consideration.
REF-REPL Output	The command line execution output for example Python console output.
REF-Github Release	The release of the build published on GitHub website.

REF-Test Case Execution Output	The output received after running the test case successfully.
Compilation and Execution Result (Application)	
REF-Application Project	Runtime instance of the software project. For example a running version of web app code.
REF-Application Status	
REF-Runtime Instance	The instance of the source code represented as app.
Documentation (Client Documentation)	
REF-API Documentation	Documentation of the API under discussion.
REF-Documentation	Any documentation that is being referred abstractly in the discussion.
REF-Documentation File in Other Repo	Documentation of other repository on GitHub.
REF-Release Notes	The notes detailing the features, bug fixes etc for each release of the software
REF-Sample Usecase	A use case of the feature under discussion.
REF-Usage Example	The example mentioned related to the use of the source code, especially for API projects.
Documentation (Project Documentation)	
REF-Blog Link	A link to the external web page, specifically blogs.
REF-Contributor License Agreement	CLA agreement requirement in the software project.
REF-Feature	The software requirement that is being developed and under discussion.
REF-License	The license given to the software project for sharing the source code.
REF-Use Case	The use case of the feature under developmennt or discussion.
Documentation (Reference Documentation)	

REF-Dependent Library Web Page	The official web page of Dependent libraries.
REF-External Web page	Any external web resource.
REF-GitHub Help Webpage	GitHub help and support web pages.
REF-Language Specification	The documentation which provides the rules and standards of the programming language.
REF-Specification	Any standards document that tells the rules of a language or API.
REF-StackOverflow Thread	The discussion forum for developers to question and answer about issues.
REF-Third-Party Website	Any website link
REF-Web Page	A resource which is being accessed by a URL on Internet.
Suggested Referents	
REF-Suggested Code Snippet	A code snippet mentioned in the discussion as suggestion.
REF-Suggested Example	An example mentioned in the discussion as a suggestion.
REF-Suggested Fix	An alternative fix mentioned in the discussion as a suggestion.
REF-Suggested Fix in previous comment	An alternative fix mentioned in the previous comment of discussion as a suggestion.
REF-Suggested Issue	An issue mentioned in the discussion as a suggestion.
REF-Suggested LOC	A Line of code mentioned in the discussion as a suggestion.
REF-Suggested PR	A PR mentioned in the discussion as a suggestion.
REF-Suggested Release Version	A release version mentioned in the discussion as a suggestion.
REF-Suggested Variable	A variable mentioned in the discussion as a suggestion.

REF-Suggested Variable Value	The value of a variable mentioned in the discussion as a suggestion.

Appendix C

Codebook for Expression Types

This chapter is the collection of 33 codes generated for referent types identified in the PR discussions. These codes are organised by their referent type and sub-category.

Table C.1: Expression Codes arranged by Expression Type (Sub-category) from Taxonomy Table 3.3

Codes	*Description*
Named Expression (Unique Identifier)	
EXP-By @Username	Name of the user mentioned using @-mention
EXP-By Abbreviation	Use of Abbreviation for a referent.
EXP-By Commit ID and Message	Using commit ID and message.
EXP-By Email	Using Email address of the referent.
EXP-By File Extension	File Extension used in the referent expression.
EXP-By Filepath	Using only file path (location) of the referent.
EXP-By Function Name Call Syntax	Function name with calling syntax, for example ”function()”
EXP-By Function Parameter Type	Function Parameter Types like int, double etc, used.
EXP-By ID#	Unique ID given to the referent.

EXP-By Name	Name of the referent.
Named Expression (Generic)	
EXP-By Pronoun Word	Using pronoun words like it, this, that etc.
EXP-By Collective Noun Word	Word used to refer to the collective referent, for example, issue, PR etc.
EXP-By Common Noun or Name	Using commonly used term for the referent. For example, ”the issue” etc.
EXP-By Demonstrative Adjective	Using demonstrative adjective, for example, ”this PR”, ”my issue” etc.
Content Based Expression (Descriptive)	
EXP-By Descriptive Name	Using Description to give a name to the referent.
EXP-By Descriptive Short Phrase	Using longer description to give an identifier to the referent.
EXP-By Descriptive Title	Using descriptive title for referent for example, for a feature or issue.
Content Based Expression (Verbatim)	
EXP-By GIF	Graphical animated image used for the referent.
EXP-By Image	Image or screenshot of the referent.
EXP-By LOC	A complete line of code verbatim.
EXP-By Verbatim Code	Exact copy of the code.
EXP-By Verbatim Copy	Exact copy of the textual referent like errors, log trace etc.
EXP-By Verbatim Text	Exact copy of the textual referent like comments etc.
Locative Expression (Absolute)	
EXP-By Line #	Line number at which referent is located, typically for source code.
EXP-By Named URL Link	Using a URL link with a name.
EXP-By URL	Link Using a URL link only.
Locative Expression (Relative)	
EXP-By index number	Number given in the footnote.

EXP-By Using above word	Keyword ”above” to refer to the referent.
Temporal Expression (Absolute)	
EXP-By Verbatim Versioning	Proper semantic versioning to specify Version number.
EXP-By Version # with specifc range	Version number with specific versioning range, for example, Butterknife 2.3.x - Butterknife 3.0.0
Temporal Expression (Relative)	
EXP-By old new last latest etc keywords	using keywords like old, new, last, latest to describe the referent properties like versions.
EXP-By Version # and vague range	Version number with loose range for example, Butterknife 7 and older versions.
Compound Expression	
EXP-By Filename + Extension	File extension and file name both present for the referent.
EXP-By Filename + Extension + Filepath	Filename, file extension and location(path) used for the referent.
EXP-By Name + Version	Using the name and version of the referent.

Appendix D

Topic Modeling Result

Table D.1: List of 29 topics with their top 20 keywords as identified by Topic Modeling algorithm.

Topic Name	Top Keywords
Web Backend Related	ruby rails code style linter chat static php static-analysis bot tool slack formatter analysis coding eslint facebook activerecord javascript messaging
Programming Languages	language programming javascript compiler python book scala functional learn emulator haskell design library programming-language swift arm cpp guide code functional-programming
CMS	php git github laravel documentation repository generator package wordpress cms code official python api management markdown static version blog site
Web Frontend Libraries	javascript css html jquery editor markdown plugin text library pdf responsive wysiwyg font svg beautiful create editing simple sass web
Gaming Engines & Libraries	game engine c-plus-plus webgl opengl javascript graphics library games cross-platform open game-engine cpp source open-source canvas vulkan rendering gamedev game-development

Topic Name	Top Keywords
Data Science	learning python machine-learning tensorflow deep-learning deep machine pytorch neural neural-network nlp data-science computer-vision models keras natural-language-processing artificial-intelligence detection implementation jupyter
Web Frontend Frameworks	vue bootstrap angular design material vuejs components vue.js admin based theme javascript webpack material-design template component css ui-kit html framework
Platform	terminal cli shell command-line command line bash python tool zsh console linux git interface files vim interactive utility environment windows
Cloud	java apache serverless aws cloud spring distributed lambda jvm spark framework kafka mirror scala platform mqtt microservices messaging reactive hadoop
Database	database sql mysql postgresql sqlite golang orm redis data mongodb postgres distributed driver key-value iot nosql realtime high-performance migrations store
Documentation	awesome list security curated awesome-list python resources collection software libraries scanner tools analysis golang frameworks framework wifi reverse-engineering security-tools hacking
DevOps	docker kubernetes containers aws container applications api golang swagger tool management openapi service infrastructure system security microservices cloud devops services
Libraries	framework web fast library simple python javascript easy performance api application lightweight php applications server powerful building java high http
Web Frontend related	image javascript images library jquery plugin video support simple camera lightweight gif slider android html processing esp responsive multiple file

Topic Name	Top Keywords
Distributed Computing	grid data library task job queue drag list distributed drop drag-and-drop lists utility based scheduler cron tasks jobs background sort
Data Visualization	data visualization javascript python monitoring library svg algorithms charts algorithm metrics graph chart canvas interview interactive structures dashboard analytics data-visualization
Web API	python library api crawler twitter instagram bitcoin written trading facebook social google flask telegram bot scraping profiler wrapper asyncio humans
Multimedia	vim audio video player music browser firefox webrtc streaming emacs chrome youtube ide extension ffmpeg neovim based editor html media
.NET Libraries	source open net platform free core dotnet c-sharp web framework open-source dotnet-core based system ecommerce csharp code software built community
Network	http server client proxy library golang websocket protocol networking php grpc security logging websockets https email fast tls request oauth
Mobile Frameworks	android ios mobile apps app framework wechat web library code ethereum deprecated cross-platform project sdk development tools flutter native devices
Web Backend Related	javascript nodejs nodejs node browser typescript graphql api json parser express library implementation module validation client schema standard middleware modules
Platform	windows linux macos electron desktop cross-platform mac gui manager app editor package c-plus-plus visual window studio application code wpf platform
Java	android java kotlin library rxjava google app spreadsheet compression notifications files write excel json push mvp reading clean android-library maps

Topic Name	Top Keywords
Android Development	android library view ios animation easy material app bar display animations calendar design menu date simple navigation custom views easily
Distributed File Sharing Related	file client search bittorrent storage elasticsearch privacy data system decentralized distributed filesystem files cloud web encryption backup self-hosted requests google
iOS Development	swift ios objective-c cocoapods macos framework written xcode carthage tvos simple library iphone mac data replacement app animation customizable notification
Front-End Frameworks	react javascript native redux component react-native components webpack angular web apps typescript reactjs forms built framework state universal boilerplate build
Testing Frameworks	testing test automation javascript tests framework tool nodejs tdd continuous mock chrome bdd continuous-integration mocking run unit headless coverage jenkins