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

ASHISH CHOPRA, University of British Columbia, Canada
MORGAN MO, University of British Columbia, Canada
SAMUEL DODSON, University at Buffalo, USA
IVAN BESCHASTNIKH, University of British Columbia, Canada
SIDNEY S. FELS, University of British Columbia, Canada
DONGWOOK YOON, University of British Columbia, Canada

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 ~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. Referencing source code elements was prevalent but the authoring interface lacks support for it. Three classes of contextual factors influence referencing behaviors: referent type, discussion thread, and project attributes. Referencing patterns may indicate PR outcomes (e.g., merged PRs frequently reference issues, users, and tests). We conclude with design implications to support more effective referencing in PR discussion interfaces.

CCS Concepts: • **Human-centered computing → Empirical studies in collaborative and social computing**; • **Software and its engineering → Collaboration in software development**.

Additional Key Words and Phrases: Pull Request, Reference, Software Development, Code Review, Taxonomy, Qualitative Content Analysis

ACM Reference Format:
Ashish Chopra, Morgan Mo, Samuel Dodson, Ivan Beschastnikh, Sidney S. Fels, and Dongwook Yoon. 2021. "@alex, this fixes #9": Analysis of Referencing Patterns in Pull Request Discussions. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 385 (October 2021), 25 pages. https://doi.org/10.1145/3479529

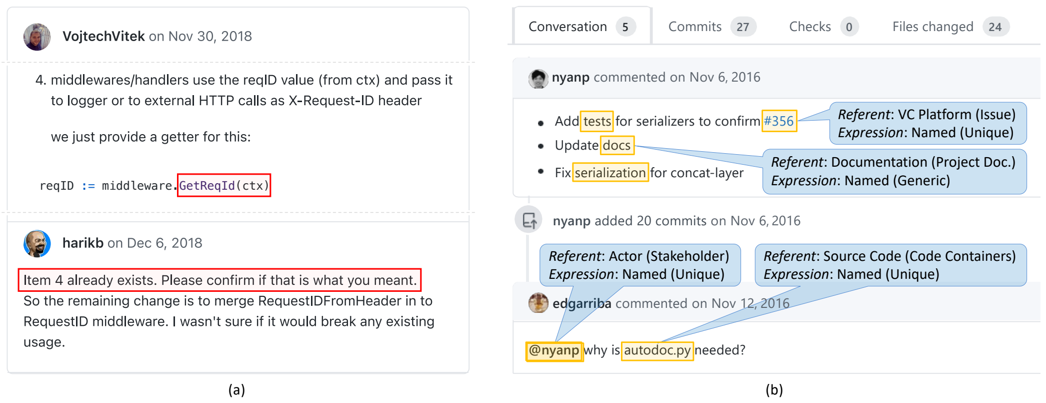
1 INTRODUCTION

Making a pull request (PR) is the primary method for software development teams to propose and review changes in code repositories [21]. GitHub reported that more than 87 million PRs were created on its platform between 2018 and 2019 [1]. 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

Authors' addresses: Ashish Chopra, ashish22@student.ubc.ca, University of British Columbia, Canada; Morgan Mo, yang.mo@alumni.ubc.ca, University of British Columbia, Canada; Samuel Dodson, smdodson@buffalo.edu, University at Buffalo, USA; Ivan Beschastnikh, bestchai@cs.ubc.ca, University of British Columbia, Canada; Sidney S. Fels, ssfels@ece.ubc.ca, University of British Columbia, Canada; Dongwook Yoon, yoon@cs.ubc.ca, University of British Columbia, Canada.

2573-0142/2021/10-ART385 $15.00
https://doi.org/10.1145/3479529

[21]. 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 [4, 14, 47]. In this paper, 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.


VojtechVitek on Nov 30, 2018

4. middlewares/handlers use the reqID value (from ctx) and pass it to logger or to external HTTP calls as X-Request-ID header

we just provide a getter for this:

```
reqID := middleware.GetReqId(ctx)
```

harikb on Dec 6, 2018

Item 4 already exists. Please confirm if that is what you meant. So the remaining change is to merge RequestIDFromHeader in to RequestID middleware. I wasn't sure if it would break any existing usage.

nyanp commented on Nov 6, 2016

- Add tests for serializers to confirm #356
- Update docs
- Fix serialization for concat-layer

nyanp added 20 commits on Nov 6, 2016

edgarriba commented on Nov 12, 2016

@nyanp why is autodoc.py needed?

(a)
Conversation 5
Commits 27
Checks 0
Files changed 24
Referent: VC Platform (Issue) *Expression*: Named (Unique)
Referent: Documentation (Project Doc.) *Expression*: Named (Generic)
Referent: Actor (Stakeholder) *Expression*: Named (Unique)
Referent: Source Code (Code Containers) *Expression*: Named (Unique)
(b)

Fig. 1. (a) An example reference breakdown. The PR reviewer (VojtechVitek) proposed a new function with the name "getReqID()", but another function with that same name already exists. The homonym confused the PR author (harikb) and delayed the review process. The red box indicates the point of reference breakdown, and the grey dashed line indicates content that was not shown in this figure for brevity. (b) An illustrated example of coded references. We assigned two codes (shown in blue boxes) to each reference (highlighted in yellow boxes). For each highlighted reference, the type and sub-category of the referent and expression is mentioned as "Type (Sub-category)" from the taxonomy tables. See Table 1 and Table 2 for the full list of referents and expressions.

Referencing information objects is a communication device that frequently serves essential purposes in PRs. Linking external resources, such as documentation and forum threads, is important in justifying proposed changes and improving documentation of the pull request [59]. Linking issues helps identify complex bugs across projects and redundant issues [32]. @-mentioning is a popular mechanism for inviting the attention of specific collaborators, delegating tasks, and distributing information to them [21, 60]. Conversely, a breakdown of reference can delay or impoverish the code review process. For example, in Figure 1 (a), a vague reference to a homonym (i.e., "getReqID()"s) without explicitly resolving the referent caused confusion and delay. Theories of grounding in communication explain that referential behaviors are crucial for establishing shared understanding [10, 11], and PR discussion is no exception.

PR is the primary mode for contributing code in collaborative software projects. Understanding referencing practices in PRs, despite their importance, is a relevant but new topic for CSCW. On the one hand, collaborative software development has been regarded as the quintessential context of CSCW. Several foundational studies in the field grew out of the investigations of software teams [24, 46], and such interests continue in more recent studies [7, 16, 48, 51, 56]. On the other hand, CSCW and HCI researchers have investigated referencing in online communications in different task contexts, including educational discussion boards [63], remote collaborative work [20], and video-based comment threads [9, 57]. The confluence of these two trends suggests the need for investigating 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 better tool support for PR discussions.

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, a reference is 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 [21, 22, 50, 52, 58]. For example, Zhang et al. [60] found that references to stakeholders through @-mentions impacts the amount of time a PR is processed. By identifying *what* and *how* people make references in PR, we aim 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 Bitbucket, GitHub, Gitlab, and SourceForge. The GitHub PR interface, for example, supports referencing platform-specific entities, such as commits, issues, PRs, and users, in addition to basic URL-embedding. However, it is not known how well existing PR interfaces support people's referencing behaviors or how to improve them. Beyond the context of software engineering, researchers have envisaged novel tools for better supporting referencing [9, 20, 37, 57]. Similarly, understanding referencing practices in PRs is the first step in designing better tools. In this paper, 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 in this paper:

(1) What information objects do stakeholders refer to in PR discussions? How do they communicate these references?
(2) How well do existing interfaces support the referential practices of users?
(3) What contextual factors influence 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, as illustrated in Figure 1 (b). 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, such as variables and functions, are the most frequently referenced type of information; however, these did *not* make up the majority of all references. Surprisingly, we found that the GitHub referencing interface has limited support for referencing source code. We also found that stakeholders have different referential practices, such as nested pointing, when it comes to internal 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 open or closed ones.

The contributions of this study are four-fold:

(1) empirical findings from 450 PRs, identifying the most commonly used reference types; how contextual factors, such as project attributes, discussion thread, and referent type, influence referential behaviors,
(2) two reference oriented taxonomies: one for what information is referenced (referent type) and another for how this is communicated (referential expressions),
(3) design implications for PR discussion interfaces that are enhanced with type-based referencing based on our analysis, and
(4) an annotated dataset of ~7K references, from 450 public PR discussions on GitHub.

2 PREVIOUS WORK

2.1 Common grounding through referencing

Grounding in communication is when people share knowledge and beliefs and is important for effective and efficient collaboration [11]. 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 discussants to identify and understand what is being referenced through their shared knowledge and beliefs [10]. For example, demonstrative references (this, these, that, those) with more than one potential referent can be de-referenced when common ground has been established [12]. Consequently, establishing referential identity, i.e., correctly identifying the object of a reference and the ability to de-reference it, is a crucial aspect of communication [41].

Previous research has looked into referencing in different kinds of collaborative platforms [8, 9, 30, 57]. Chua et al. [9] 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 [8] helps to make and resolve references to physical objects. Yarmand et al. [57] 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 fresh insights to better understand referencing behaviors specific to source code PR discussions and novel design implications for enhancing referencing to information resources in PR discussions.

2.2 Source code review

Effective communication has a significant impact on the software development lifecycle. A quality modern code review process is important for ensuring the long-term maintainability of the code base [40]. 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. [26] found that developers still expect asynchronous code reviews to be clear and thorough. Efstathiou et al. [19] 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. [55] 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 [17, 26, 55], little attention has been paid to the references used in the comments, which is a key for establishing a common ground [11].

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 [21, 22, 50, 52, 58]. Gousios et al. [21] 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. [22] 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. [52] found the strength of connection between a PR contributor and the stakeholders conducting the code review is important. Soares et al. [50] 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. 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 reference 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 [6, 18, 33, 42, 47]. 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. [47] 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. [18] 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 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 [5, 27, 36, 57].

The research literature has had success creating taxonomies in order to describe the communicative behaviors of a wide range of communities. These taxonomies, overall, help to explain what rhetorical roles information objects play. However, no taxonomies exist describing references in the code review process appropriate for our analysis. 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 Section 3.

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 1) and *referential expressions* (Table 2) using the research design presented in Figure 2. These lay the foundation for analysing referential behaviors in Section 3.5.

To create these taxonomies, we conducted a qualitative content analysis (QCA) [28] 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 [61], and is a useful method for generating inductive taxonomies [23, 35, 36]. For data collection, we used GitHub because it is the largest platform for open source software development. GitHub's PR feature is similar to those provided by other platforms, such as Bitbucket, Gitlab, and SourceForge. Similarity in PR features across these platforms increases the generalizability of our taxonomies. We conducted the QCA in three steps:

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

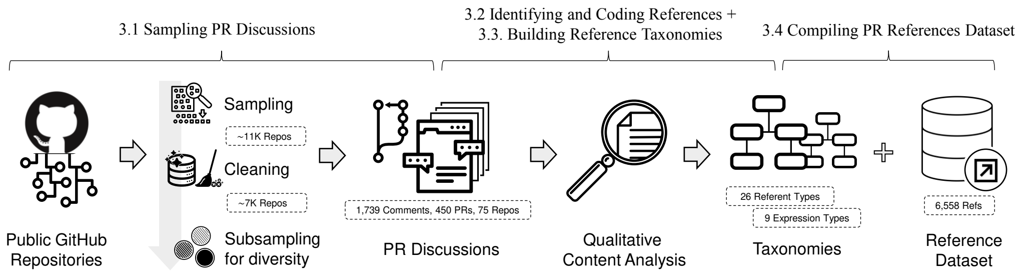

3.1 Sampling PR Discussions
3.2 Identifying and Coding References +
3.3. Building Reference Taxonomies
3.4 Compiling PR References Dataset
Public GitHub Repositories
Sampling
~11K Repos
Cleaning
~7K Repos
Subsampling for diversity
1,739 Comments, 450 PRs, 75 Repos
PR Discussions
Qualitative Content Analysis
26 Referent Types
9 Expression Types
Taxonomies
6,558 Refs
Reference Dataset

Fig. 2. A flow diagram representing our research design for compiling an annotated dataset of ~7K references from 450 public PRs from GitHub. The dataset is used to analyze referential patterns in PR discussions.

(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, by accounting for five project-based attributes. We used four attributes provided by GitHub (programming language, star count, number of contributors, and number of PRs) and identified the application domain of each project ourselves. The data snapshot was taken on 18 November, 2019.

Our sampling involved five steps. First, we scraped ~11K most popular repositories from GitHub, based on their star count. Second, we filtered out ~4K projects that were either 1) not software-based, 2) were missing descriptions or keywords, or 3) had fewer than six PRs, leaving us with ~7K repositories. Third, we identified the application domain of these ~7K projects using Latent Dirichlet Allocation topic modelling. Fourth, we used Nagappan et al.'s [43] *sample coverage* measure to create a representative sample of 75 distinct repositories from the population of ~7K repositories, accounting for the five diversity criteria. Our sample of 75 repositories covered 40.8% of the total variance. Fifth, we randomly selected two merged, two closed, and two open PRs from each of the 75 repositories to diversify over the PR statuses, resulting in the dataset of 450 PRs (2 PRs × 3 Statuses × 75 Repos = 450 PRs).

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.

Two of the authors performed QCA, and the process was discussed among all authors 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 Section 4.

The coders worked together on 26% of the dataset for training. During the training phase, a code book was co-authored by them in three iterations. We developed codes for both referent type and referential expression until we reached saturation. We refined the codes in this phase though discussions 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 1 (b), where each reference is highlighted and is assigned two codes: one for referent type and one for expression. It took a total of 560 hours per coder to code 450 PR discussion threads, of which the training phase took 40% of the time. The code book is available in the supplementary materials. 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 [13] were 0.76 for the Referent Taxonomy and 0.72 for the Expression Taxonomy, indicating a strong agreement between the coders [39]. 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 1). The process of building taxonomies took ~400 hours in total over four iterations by the lead author who discussed with all other authors 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, and so on. The other categories in the taxonomy, such as ACTOR, VC PLATFORM, COMPILATION AND EXECUTION RESULT, DOCUMENTATION, and DEV TOOL AND ENVIRONMENT, are comprised of non-source code entities. In the last iteration, we added several new codes generated during the last batch of coding. Similarly, 33 expression codes were classified into four top-level categories and nine sub-categories in our expression taxonomy (Table 2). The full directory of referent and expression codes of these taxonomies is provided in the supplementary materials.

3.4 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 publicly available at Zenodo [3]. 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. In total, it took 360 hours to develop, test, and execute the data extraction pipeline for compiling the dataset from the coded PR threads. We used this dataset for our exploratory data analysis of referential behaviors, which we present in Section 4. The R scripts for data analysis are also available at Zenodo [3].

3.5 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 [21,

Table 1. Taxonomy of referent types, with examples.

Category	Sub-category	Definition	Example(s) from Dataset
SOURCE CODE	*Code Container*	A container unit which contains the source code, typically a file.	Only change in abstractclientbase.js is this...
	Code Element	The building blocks that make up the source code, such as symbols, keywords and various constructs.	.doesNotThrow() is currently not matching the error message...
	Input and Value	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 Library	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 discussions 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 Entity*	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 TOOL AND ENVIRONMENT	*Software Development Tool*	Tools used by people to develop the software project.	This enables some new warnings, as recommended by Xcode.
	Automation Tool	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/

22, 50, 52, 58]. 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 authors of the paper conducted an exploratory data analysis [54] to identify patterns in the references dataset. We used a mixed approach which includes inductive and deductive

Table 2. Taxonomy of expression types, with examples.

Category	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.

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 authors of the paper to reach an agreement. The two coders spent ~400 hours in total analyzing the PR references.

To explore the co-occurence 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 [53], as a pre-process 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 5a) 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 [25] (see Figure 5b). The relevant analysis of co-occurring referent types is presented in Section 4.

4 FINDINGS

We found referencing of information resources to be prevalent in PR discussions. We identified 6,558 references in 2,189 comments. 93.5% of the comments referred to at least one information entity. On average, a comment has 2.99 references (SD = 3.15 , Min = 0, Max = 59). We use "comments" to refer to the titles and threaded messages of PRs. A PR, on average, has 4.86 comments (SD = 4.33, Min = 2, Max = 54).

Referencing is a critical communication device for establishing common ground. As shown in the example in Figure 1 (a), discussants in a PR can misunderstand each other's comments due to a referencing breakdown. Having correct and unambiguous references is crucial for productive code review and discussion.

The following sections document our findings on how well current interfaces support referencing in PRs, which contextual factors influence referencing behaviors, and how referencing patterns might indicate different PR outcomes.

Table 3. Distribution of references created using different referencing mechanisms

Referent Type	Auto-linked	URL-Linked	Plain Text	Total	Excluding Generic	
					Plain Text	Total
SOURCE CODE	0	76	2,910	2,986	2,203	2,279
VC PLATFORM	368	56	1,309	1,733	461	885
ACTOR	332	30	245	607	166	528
DOCUMENTATION	0	156	388	544	249	405
DEV TOOLS AND ENVIRONMENT	0	4	360	364	318	322
COMPILATION AND EXECUTION RESULT	0	11	313	324	224	235
Total	**700**	**333**	**5,525**	**6,558**	**3,621**	**4,654**

4.1 Empirical Accounts of Referencing Support on GitHub

In GitHub, there are three ways for PR discussants to refer to information resources. The auto-linking features [2] detect references prefixed with special characters (e.g., “#456”, “@reisenberger”) and then automatically convert them into links. However, auto-linking is only available for a handful of the referent types we identified in our taxonomy, specifically VC PLATFORM (*issues*, *PR*, and *commit*) and ACTOR (*stakeholder*). The GitHub interface also detects web addresses and convert them into links. We refer to these as URL-links to differentiate them from auto-links. For information entities without auto-linking or URL-linking support, users must write plain text references (e.g., “paperFontButton”, “flake8 issues”, “IE 11”). In this section, we focus on providing empirical findings that underpin the extent to which existing interface features support different types of references or leave certain reference types unsupported.

Among all reference instances in our dataset, the majority (84.2%, 5,525 out of 6,558) was in plain text, while only 15.8% were either auto-links or URL-links (Table 3). The plain text references were over represented, because, discussants tend to use *generic* terms such as pronouns, adjectives, common nouns, and collective nouns (e.g., “issues”, “unit tests”, “this PR”, “the function”, “it”, etc.) that are all in the plain text format when referring to a collective class of referents or an established reference that appeared early in the thread. These references do not establish any referential identity. Instead, they assume that the referential identity of the information resource is already established. To count direct, specific references, 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 2). In the remaining 4,654 references, the proportion of plain text went down, as shown in the “Excluding Generic” columns of Table 3.

4.1.1 *Auto- and URL-linking are frequently used when available.*

GitHub’s auto-links and URL-links are frequently used when referencing a supported referent type. The majority of ACTOR (62.9%, 332 out of 528) and a significant portion of VC PLATFORM (41.5%, 368 out of 885) references were made with the auto-linking features (Table 3). URL linking was popular for linking to DOCUMENTATION. However, plain text was used for the majority of all other referent types, especially SOURCE CODE (96.7%, 2,203 out of 2,279), DEV TOOL AND ENVIRONMENT (98.8%, 318 out of 322), and COMPILATION AND EXECUTION RESULT (95.7%, 224 out of 235).

Auto-linked referencing feature is most frequently used for platform-related referent types. We found that platform resources, such as *Issue*, *PR*, and *Stakeholder*, are the most frequently referenced types when using auto-linked referencing interface (see Figure 3). These information resources are native to the GitHub platform, and have built-in referencing support through the auto-linking features in the authoring interface. For example, a developer can auto-link an issue or a PR with the

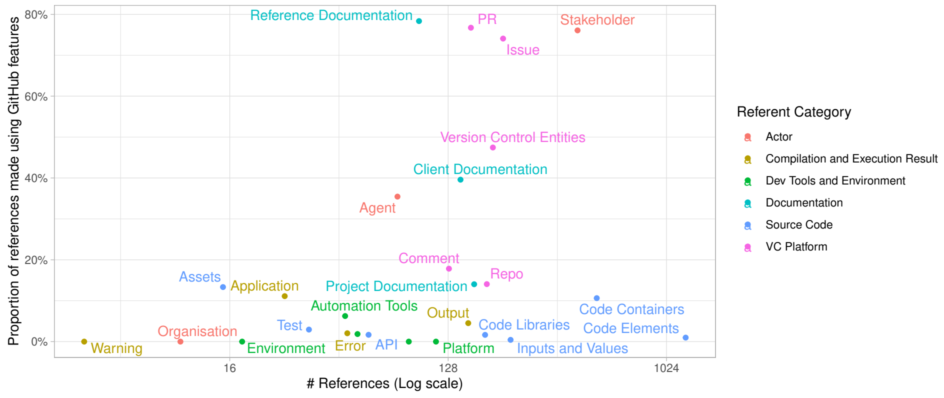


Fig. 3. Distribution of referent sub-categories, based on the number of non-generic references (x-axis) and the proportion of those references created using the GitHub auto-linking features (y-axis).

“#” character followed by the issue or PR number, as in “This fixes #9.” Similarly, a GitHub user can be referenced in a comment with the “@” character followed by a username: “@reisenberger thanks for the tag!”. The prevalent use of these references in comments is likely due to the auto-completion, which activates as soon as a user types any of the special referencing characters (# and @) within the authoring interface. This makes it easy to search for and link to references while commenting.

At present, the auto-linking features only support platform-specific entities; however, they do not provide full coverage. For example, *Repo*s and *comments* in Issues and PRs are unsupported. 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, a PR submitter referenced another repository in his comment through a URL link: “I made this from combining parts of the ipython/ipython and jupyter/jupyter Dockerfiles”.

Stakeholders rarely link URLs in their comments. We found that 7.2% of references (333 out of 4,654) in our dataset are URLs (Table 3). *Reference Documentation* outside GitHub, such as external web pages and forum threads, are primarily referenced by URL-linking (see Reference Documentation in Figure 3). However, the other referent types used it sparingly. 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 unnecessary for resources that are already a part of the project, such as project documentation, readme, bots, and source code files. These are primarily mentioned in plain text.

4.1.2 *Most referent types are expressed in plain text.*

We found that the majority of references (77.8%, 3,621 out of 4,654) were written in plain text without a link (Table 3). The higher number of plain text references in PR discussions may be due to the lack of support for 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 in the referencing interface and is discussed primarily in plain text (Table 3). 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 sentence like “The fontSize for paperFontButton is 14”. Similarly, a PR in a UI project has a comment “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 are explained in long text. Writing and

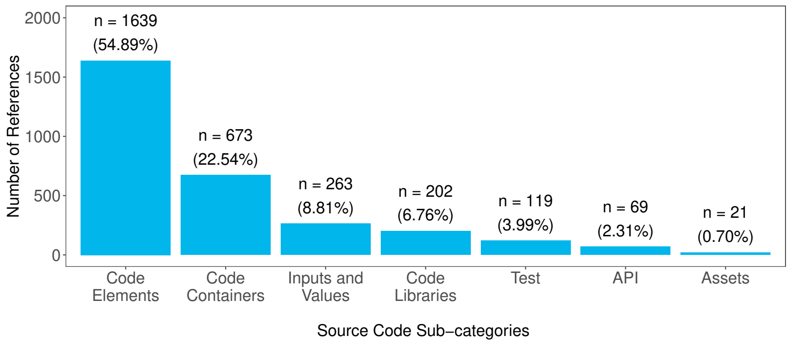


Fig. 4. Distribution of SOURCE CODE referent type by sub-categories.

interpreting comments becomes even more challenging when the discussion involves *Application* referents, such as runtime instances of source code: "Click on menu1, then menu2, then menu1 again → JS error is occurred." Here, UI navigation is described in writing to reproduce the issue which is difficult to dereference.

On the other hand, plain text is an appropriate choice of expression for some referent types, such as DEV TOOL AND ENVIRONMENT, COMPILATION AND EXECUTION RESULT, and DOCUMENTATION. For example, *Software Development Tool*s, including automation tools, code editors, and browsers, are mainly presented in text: "Fixes issue with m.route in IE 11, caused by this line." References to tools (e.g., "IE 11", "chrome", "windows", "android", "JVM") are commonly used in software engineering discussions and can be easily understood in plain text.

We found that ~18.8% of plain text references express the referents verbatim. For example, compilation/runtime *Error*, and console *Output* are generally provided as-is by copying the output log into comments (Figure 3). This is likely because *Error*s cannot be easily understood by other reference handles, such as error codes. Therefore, complete logs are useful for communicating the error in the discussion. However, producing verbatim references by copying and pasting is time-consuming. Also, referencing and dereferencing the verbatim reference in subsequent comments takes effort.

On the other hand, some plain text references use descriptive phrases to reference information, including *API*, *Warning*, and *Test Case* (Figure 3). For example, in one of the PR comments, the reviewer suggested a new API: "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". This was picked up in the thread and used as a reference handle while discussing the use case scenario by other discussants. We observed similar practices for *Test*, *Warning* and *Issue*, which are given short descriptive titles (e.g., "coroutine test", "peer dependency warnings", "flake8 issues", "typescript bug") that serve as reference handles throughout the thread.

4.1.3 Source code is frequently discussed in PR discussions, but is unsupported by auto-linking.

There are many references to source code in PR discussions despite being unsupported by the authoring interface. We found that 49.0% of all references in our dataset (2,279 of 4,654) are to SOURCE CODE referent types (Table 3). *Code Element*s and *Code Container*s are the most prevalent of these (54.9% and 22.5% respectively, see Figure 4). However, only 3.3% of total SOURCE CODE references were created using the GitHub referencing interface, primarily by linking URLs (Table 3). We present referencing patterns in the two major source code types below.

Table 4. Distribution of references to Source Code sub-categories by referencing mechanism (N = 2,279). GitHub does not support auto-linking for the Source Code type references.

Sub-category	**URL-linked**	**Plain Text**
API	1 (1.6%)	59 (98.3%)
Asset	2 (13.3%)	13 (86.7%)
Code Container	56 (10.6%)	471 (89.4%)
Code Element	12 (1.0%)	1,217 (99%)
Code Library	3 (1.6%)	179 (98.3%)
Input and Value	1 (<1%)	231 (99.5%)
Test	1 (2.9%)	33 (97.0%)
Total	76 (3.3%)	2,203 (96.7%)

Table 5. Distribution of PRs by SOURCE CODE and non-source code referents.

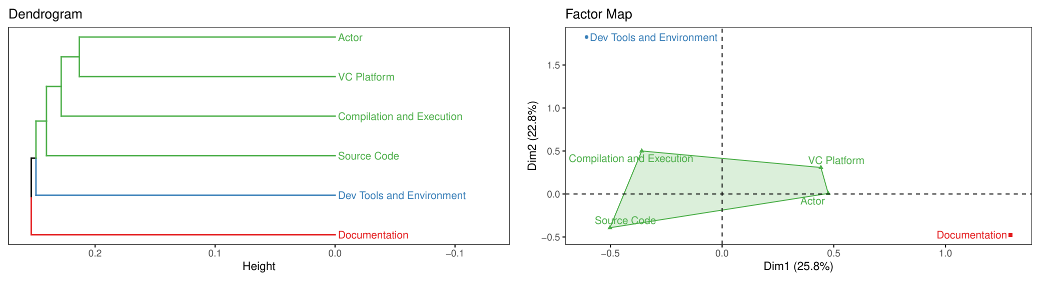
Referent Type(s)	**Count of PRs**
SOURCE CODE and Non-source code	356 (79.1%)
Non-source code only	60 (13.3%)
SOURCE CODE only	34 (7.5%)
Total	450 (100%)

Plain text-based referencing to Code Elements can be tedious. Code Element (e.g., variable, function, and class) is the most frequently referenced (54.9%) SOURCE CODE type in PR discussions (Figure 4). 99% of *Code Element* references are made using plain text. Since there is no direct URL for code constructs, such as variables, functions, and classes, they are referred to manually (Table 4). For example, a PR contributor described the content of the PR by referencing functions with plain text: "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." The contributor also referenced classes similarly: "MetricsSpan, CounterSpan and TimingSpan controversial, this PR only implements a generic Span that is a timing one". Referencing these resources becomes challenging in repositories containing code elements with the same names located in different files. Often, developers try to resolve this ambiguity by mentioning the fully qualified name of the function or class in text, for example: "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". This helps to disambiguate the reference, but is burdensome and error-prone. We observed developers working around this by inserting URLs for the lines of code where the function or class is located in the file. This involves context switching, requiring the author to leave the discussion interface in order to identify and copy the link location. Dereferencing source code references in plain text is also tedious for readers of the PR discussion, and requires navigating to different web pages to locate where the referent is situated.

*Developers reference code containers by writing text descriptions, which can be difficult for viewers to dereference. Code Container*s, such as code files, folders, and lines of code, are the second-most referenced type (22.5%) of SOURCE CODE (Figure 4). The GitHub interface allows developers to reference lines of code and code files with URLs. However, we found that the majority of *Code Container* references (89.4%) are made in plain text, with only 56 out of 527 (10.6%) created using URL-links (Table 4). For example, a discussant mentioned a line of code by specifying the line numbers in plain text: "Can we wrap the try started on line 97 in a single try/catch", and similarly referenced a file: "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 establish a mutual understanding of the context.

4.2 Contextual factors that shape the reference use

The references in PR discussions are indexical in nature, and are understood by knowing the associated *context* of the discussion. The context shapes how these references are made. We present a qualitative description of how references are shaped by three classes of contextual factors: referent, discussion thread, project attributes. These findings add to the factors known to affect general information practices, as identified by Courtright [15].



(a) (b)

Fig. 5. The output generated by the correspondence analysis and hierarchical clustering: (a) Dendrogram on principal components on the top-level referent types (b) Factor map on principal components on the top-level referent types.

4.2.1 Referent influences expression.

The type of information resource being referenced influences the use of references in PRs: certain types of references often co-occur. For instance, we found that SOURCE CODE types frequently co-occur with non-source code types (Table 5). Discussants leverage these references to motivate and specify the need for the suggested code changes. The cluster analysis shows that some referent types are more relevant to SOURCE CODE than the other categories. SOURCE CODE references frequently co-occur with the ACTOR, VC PLATFORM, and COMPILATION AND EXECUTION RESULT referent types, but not with DEV TOOL AND ENVIRONMENT or DOCUMENTATION (see Figure 5b). Below we present our findings on how referent type shapes the reference usage.

References to documentation frequently include additional information to increase 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 frequently expressed as plain text because nested sections of information resource might not be locatable by a URL. Sometimes, developers combine links with plain text providing additional information about the location of the referent. This saves the other discussants' time in locating the information and also prevents misunderstanding.

Developers tend to express compilation errors verbatim and warnings descriptively. COMPILATION AND EXECUTION RESULT referent types such as *Error* 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 6). 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.

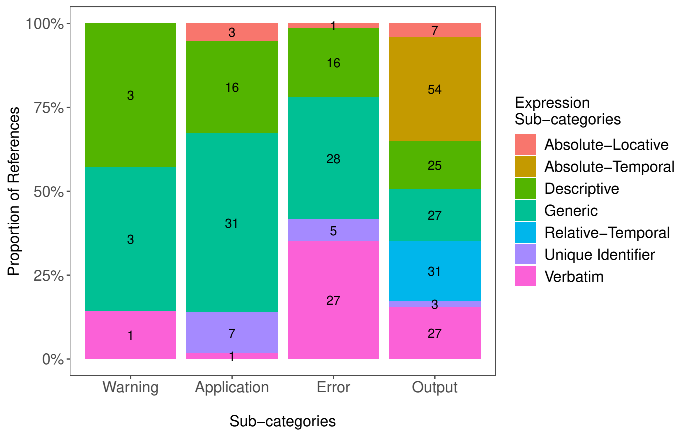


Fig. 6. Distribution of Expression Sub-categories for COMPILATION AND EXECUTION RESULT.

Developers tend to use temporal information for Build Release and Execution Output. Many referent types, including many from the COMPILATION AND EXECUTION RESULT and VC PLATFORM are frequently expressed using *temporal* descriptors, such as their version number. 85 out of 174 (48.9%) references to *Output* artifacts, such as build releases, explicitly employ *temporal* expressions while the other referent types did it seldom. 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. Knowing temporal information of these referent types is necessary for reviewers to gauge the impact on the code base and helps in prioritizing PRs.

4.2.2 *Discussion thread influences referencing.*

Each PR discussion thread varies by its topic, length, discourse complexity, and discussion stage. Reference use may indicate such attributes of PR discussion threads. Below we elaborate on our novel findings regarding how the nature of a discussion thread shapes referencing behaviors.

Developers frequently use generic terms to reference information scoped within a discussion thread. We found that 30.2% of the references in our dataset are expressed using *generic* terms, including pronouns, adjectives, and collective nouns (Figure 7). These references 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 "EnhancedFactoryData", 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, and there is no need to reference it by its name again. 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

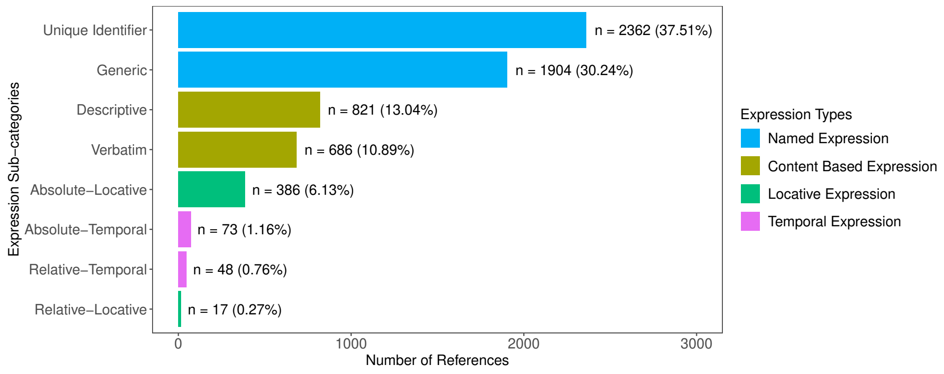


Fig. 7. Distribution of Expression Sub-categories.

referent, for example, referring to the "contributors" instead of a specific user, collective nouns are used as generic expressions. References with generic terms may 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.

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 *Issue* and *PR*. 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 5). 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.

4.2.3 *Project attributes influence referencing.*

We identified instances where the characteristics of projects shape referencing behaviors of stakeholders. Software projects vary by goal, programming language, age, size, team composition, etc. It is a reasonable conjecture that these factors can attribute to and influence what stakeholders mention in PRs and how those references are communicated. Two patterns were apparent in our qualitative analysis.

Referring to UI elements is common in mobile and web-development projects but difficult. Android, iOS, and web-based development projects frequently reference a substantial number of UI elements, including buttons, menus, and margins, as well as how UI elements animate on screen and respond to input. 6.3% of references in these projects refer to UI elements as compared to other projects that contain only 0.7% of UI-related references. Despite their prevalence, GitHub does not support an interface feature to mention UI elements specifically. Hence, stakeholders exerted more effort to express these references, such as writing descriptive names or phrases, or linking screenshot images. For example, in one PR discussion participants referred to a button in the GUI by its screen name: "show more button". Also, to express transient information (e.g., UI effects, states, and transitions), which cannot be pointed to easily by name, like a button in the above example, participants used GIFs or images in pairs to show before and after scenarios. This approach also adds to readers' workload decoding the reference.

Developers commonly use suggested references in projects of web-based programming languages. We identified references in PR discussions to information resources that do not yet exist. We call these *Suggested Referents*. Any referent type in our taxonomy Table 1 can be a suggested referent. We found that suggested referents are commonly used in projects using JavaScript, CSS, and PHP as their primary language. For example, in one CSS project's PR, 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. As web-based programming languages, such as CSS, offer many ways to implement the same feature in a relatively few lines of code, we suspect that the expressive nature of these languages is responsible for the pattern we observed. The increase in suggested referents also indicates that developers in such projects tend to have discussions related to alternative solutions and suggestions. Also, a larger number of suggested referent types are found in closed PRs as compared to merged PRs, implying that the discussion led to revising the submitted source code.

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 present an analysis of referencing patterns observed in merged and closed PRs, because these two statuses are conclusive while the open status is indefinite, and a decision has yet to be made. We also describe our high-level observations from open PRs (i.e., PRs that are neither closer nor merged).

4.3.1 Accepted or Rejected PRs tend to have references to the evidence which advocate their appropriateness. References to *Test*, *Agent*, and *Issue* are more frequently found in merged than closed PRs (see Figure 8). 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 correctness 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 about them due to the small sample size (N = 17).

References to *Automation Tool* and *Version Control Entity* are more than twice likely to be found in closed PRs than merged PRs (Figure 8). These pattern indicates that the discussion may have

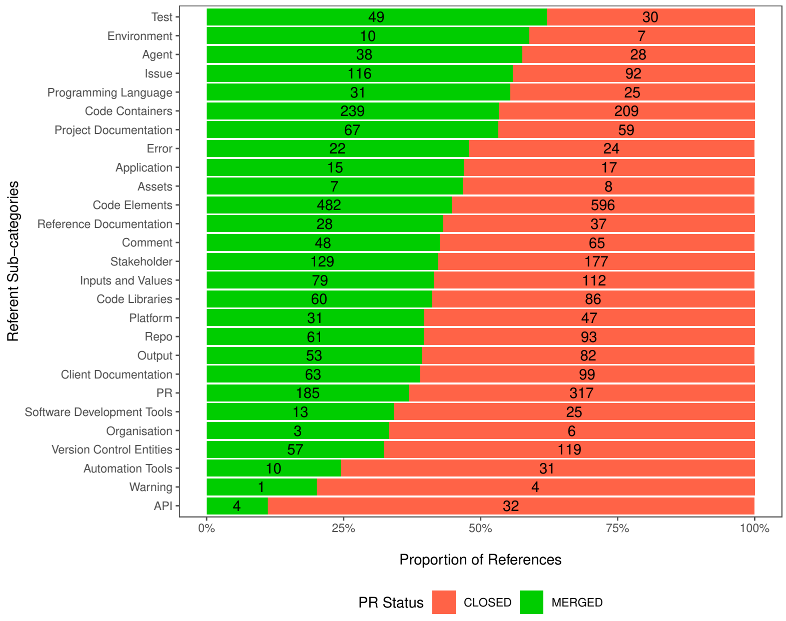


Fig. 8. Distribution of all Referent Types by PR status (Closed/Merged).

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 Tool* references in closed PR discussions may indicate the negative outcomes of a build system on a given PR [62]. Similarly, *Version Control Entity* 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 API-heavy PRs might be more likely to be closed (N = 32) than merged (N = 4), because such discussions tend to be contentious and can easily lead to rework (see Figure 8). 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 a relatively high proportion of API-level changes in the software or compatibility issues are found in open PRs (33 of 69 API instances) than the other two PR 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, one PR we considered proposed an API for client interceptors in which discussion hinged on the API's compatibility across different runtime platforms. This PR included four active discussants. The discussion ended without any decision around the time that we sampled this PR in our study. Fast forward 5 months later, and the PR was closed with a message that the proposed changes were in conflict with the long-term API goals of the project. We believe that such API-changing PRs are more likely to remain in an open state, because consensus is more difficult to reach.

Table 6. Summary of implications for tool design and future research.

	Design Implications
1	Referencing finer details in source code.
2	Referencing UI elements in PR Discussion.
3	Supporting nested referencing in comments with transclusion.
4	Supporting references within a PR discussion thread.
5	Extending GitHub's current support for auto-linking software deliverables.
	Research Opportunities
6	Exploring the use of references in different stages of discussion.
7	Exploring the relationship between referencing and PR outcomes.
8	Extending the support of referencing to discussions beyond software engineering domains.

5 DISCUSSION

This paper provides insights into developers' referencing practices in PR discussions as summarized in Table 6. We further consolidate our findings into two major takeaways for the design and research community. First, many prevalent referencing practices provide design implications for supporting richer 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 outcomes, which we elaborate on in Section 5.2 and Section 5.3. Finally, we end the chapter with a discussion of the lack of referencing support in communication occurring beyond the context of PRs.

5.1 Design implications

We identified various referencing patterns in PR discussions that are frequently used by the stakeholders but are not supported by GitHub's auto-linking features. For example, source code is a frequent object of discussion in PRs, but it is primarily discussed in plain text. Similarly, there is no support for referencing UI elements, nested content in 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 support referencing and dereferencing various underrepresented referent types through GitHub's PR discussion interface.

5.1.1 Referencing finer details in source code.

Our findings in Section 4.1.3 indicate that *Code Element*s in the SOURCE CODE category are a major part of PR discussions. However, developers are making these references manually, which is tedious and can be error-prone. Typing correct variables names or function signatures is important, because that is the only index into the source code. Although GitHub renders the code differently from the regular text in the comments, which provides a visual distinction, it does not support referencing source code within PR comments. This suggests the need for *in-situ* code editing capabilities within the PR interface. This could assist developers to reference and dereference various code elements in the source code within the repository. Like a code editor, the interface would provide content assist (i.e., code completion) to write code faster and more efficiently. When a developer types in the first few letters, it would provide a list of matching code elements (variables, functions, classes, etc.) drawn from the source code within the project. The developer could then make a selection from the list and the interface would automatically create a link to the referent.

This feature would allow developers to type code with few keystrokes while maintaining accurate and precise links. It would be even more useful when a function or variable with the same name exist in different code files, as the feature would link to the correct definition. These auto-linked code references could be dereferenced in many ways. By hovering over the reference (e.g., a function name) in the comment, the interface would display a function definition in a tooltip within the PR thread. Or, by clicking the reference, a user could navigate to the location of the referent in the source code file. The *in-situ* code-editing capabilities within the commenting interface of PR discussion would speed up coding authoring within comments and improve efficiency by reducing typos and other common mistakes.

5.1.2 *Referencing UI elements in PR Discussions.*

We found that projects with interface elements reference various resources, including UI elements, layouts, colors, and positions, in their comments. Referencing these resources requires verbose descriptions. It is even more difficult to reference information that cannot be pointed to on the UI, such as state changes, animations, and background effects. Developers interleave screenshots with text descriptions in their comments, which is tedious and time-consuming. To reproduce issues, or demonstrate the implemented changes, developers reference links to the running instance of the app on web-based IDEs such as StackBlitz[1] and 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. An emulator could run the app next to the discussion thread in the PR and allows users to access different UI elements. For example, a user could reference a button in the app by selecting the button in the emulator and dragging it into the commenting interface, which would then automatically add a link to the button in the comment. Developers could also take a screenshot of the UI within the PR interface without having to switch their context. These linked UI references in the comments could be dereferenced by clicking the link, which would load the UI in the emulator within the PR discussion.

5.1.3 *Supporting nested referencing in comments 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 [31, 45]. 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 [29, 38]. 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. [9] 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 *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", and "Timing API". Once introduced in the scope of discussion, these references are later used by others in subsequent replies. However, there is no mechanism to trace the location

[1]https://stackblitz.com/
[2]https://gitpod.io/workspaces/

where these references are created. Therefore, users who are not actively involved in the discussion need to search the entire thread to locate the definitions of these references. These references become even more difficult to decipher when one comment in the PR is cross-referenced in another PR. This suggests a need for an NLP-based 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 could use such a feature to 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 4.2.1 that developers make text references to software deliverables such as build releases, project documents in their comments during discussion. Although, there is a way to link URLs to these resources by copy-pasting them into the text, this feature has not been used effectively. It is time-consuming and requires context switching. Since GitHub's auto-linking feature already supports referencing many platform-specific entities, including Issues, PRs, and users, it would be reasonable to extended its support to other resources within projects, such as build releases and project files. To reference a build release, a user could type a prefix character to invoke quick navigation within the authoring interface. The quick navigation would provide the list of releases published by this repository on GitHub. The user could navigate the list and make a selection, and a linked reference to the release would be added to the comment. This feature could enhance a common linking feature provided by GitHub for auto-linking external resources [49]. 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 the code review process, PR discussions flow through different discussion stages. Referencing can act as a benchmark as to the current stage of the discussion. For example, the title and description provided by the submitter of 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: 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 over time. These references can orient the flow of the PR discussion in a specific direction. Therefore, it would be interesting to explore what set of references impact the PR discussion in helping it to make progress. This kind of study may also reveal commonly occurring behaviors in these conversations, as well as well-established norms that can have useful implications. For example, the kinds of references made by people to come to an agreement could inform the design of an automated bot which can facilitate the decision-making process by directly introducing, or encouraging, similar comments. Also, knowing the stage of discussion, we could build predictive models to predict when a PR has reached a conclusion. This may help to improve 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. [57] showed that referencing timestamped video snippets in comments can increase the number of "likes" and replies on YouTube.

Similarly, Liu et al. [34] found that messages that reference multimedia content on Twitter receive more re-tweets. On StackOverflow, answers with references receive more ”up” votes [44]. 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 [21, 22, 50, 52, 58]. 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. For example, in 2011, GitHub introduced auto-linking users using @-mentions the PR interface [2]. Similarly, in 2019, a custom prefix feature for creating hyperlinks to external web resource was released to enterprise-tier customers [49]. In Section 4, we only captured referencing behaviors that took place when the users were given only a type of referencing feature set. Future work could 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 could 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 references, therefore, support for other platform-entities like repositories can be included. Similarly, other studies have also found that general support for referencing is lacking in contexts beyond code review. Yarmand et al. [57], 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 [44].

6 CONCLUSION

In this paper 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 a detailed empirical account of how well referencing is supported by GitHub. We provided an analysis of the patterns of what people refer to (type) and how they refer it (expression). 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 referent type, discussion thread, and project attributes are three classes of 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 provide some more general insights about referencing behaviour in PR discussions and reflect on how this human behaviour appears in other contexts leading to design strategies for providing better tool support.

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.

7 ACKNOWLEDGEMENTS

This work was partially supported by the Huawei-UBC Joint Software Engineering Research Program, NSERC Discovery Grant, and Designing for People CREATE program.

REFERENCES

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

[2] 2020. Autolinked references and URLs - GitHub Docs. https://docs.github.com/en/github/writing-on-github/autolinked-references-and-urls Accessed Oct 29, 2020.

[3] 2021. Public Dataset: "@alex, this fixes #9": Analysis of Referencing Patterns in Pull Request Discussions. http://doi.org/10.5281/zenodo.5081029 Accessed Jul. 7, 2021.

[4] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[5] Lisa Baron, Jean Tague-Sutcliffe, Mark T Kinnucan, and Tom Carey. 1996. Labeled, typed links as cues when reading hypertext documents. *Journal of the American Society for Information Science* 47, 12 (1996), 896–908.

[6] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix?. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, 202–211.

[7] Pernille Bjørn, Morten Esbensen, Rasmus Eskild Jensen, and Stina Matthiesen. 2014. Does Distance Still Matter? Revisiting the CSCW Fundamentals on Distributed Collaboration. *ACM Trans. Comput.-Hum. Interact.* 21, 5, Article 27 (Nov. 2014), 26 pages. https://doi.org/10.1145/2670534

[8] Yuan-Chia Chang, Hao-Chuan Wang, Hung-kuo Chu, Shung-Ying Lin, and Shuo-Ping Wang. 2017. 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*. 2246–2259.

[9] Soon Hau Chua, Toni-Jan Keith Palma Monserrat, Dongwook Yoon, Juho Kim, and Shengdong Zhao. 2017. Korero: Facilitating complex referencing of visual materials in asynchronous discussion interface. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–19.

[10] Herbert Clark. 1996. *Using Language*. Cambridge University Press, Cambridge, United Kingdom.

[11] Herbert Clark and Susan E Brennan. 1991. Grounding in communication. (1991).

[12] Herbert Clark, Robert Schreuder, and Samuel Buttrick. 1983. Common ground at the understanding of demonstrative reference. *Journal of Verbal Learning and Verbal Behavior* 22, 2 (1983), 245–258.

[13] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.

[14] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2016. Findings from GitHub: methods, datasets and limitations. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 137–141.

[15] Christina Courtright. 2007. Context in information behavior research. *Annual review of information science and technology* 41, 1 (2007), 273–306.

[16] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work* (Seattle, Washington, USA) *(CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 1277–1286. https://doi.org/10.1145/2145204.2145396

[17] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2018. Communicative intention in code review questions. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 519–523.

[18] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. 2019. Confusion in Code Reviews: Reasons, Impacts, and Coping Strategies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 49–60.

[19] Vasiliki Efstathiou and Diomidis Spinellis. 2018. Code review comments: language matters. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. 69–72.

[20] Susan R Fussell, Robert E Kraut, and Jane Siegel. 2000. Coordination of Communication: Effects of Shared Visual Context on Collaborative Work. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. 21–30.

[21] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355.

[22] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1* (Florence, Italy) *(ICSE '15)*. IEEE Press, 358–368.

[23] Devon Greyson, Heather O'Brien, and Saguna Shankar. 2020. Visual Analysis of Information World Maps: An Exploration of Four Methods. *Journal of Information Science* 46, 3 (2020), 361–377.

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

[25] François Husson, Julie Josse, and Jérôme Pagès. 2010. Principal Component Methods-Hierarchical Clustering-Partitional Clustering: Why Would We Need to Choose for Visualizing Data? , 17 pages.

[26] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*. 1028–1038.

[27] Richard W Kopak. 1999. Functional link typing in hypertext. *ACM Computing Surveys (CSUR)* 31, 4es (1999), 16–es.

[28] Klaus Krippendorff. 2019. *Content Analysis: An Introduction to Its Methodology*. SAGE, Los Angeles, CA.

[29] Harald Krottmaier and Hermann A. Maurer. 2001. Transclusions in the 21st Century. *Journal of Universal Computer Science* 7, 12 (2001), 1125–1136.

[30] André Kunert, Alexander Kulik, Stephan Beck, and Bernd Froehlich. 2014. Photoportals: shared references in space and time. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 1388–1399.

[31] George Landow. 2006. *Hypertext 3.0: Critical Theory and New Media in an Era of Globalization*. Johns Hopkins University Press, Baltimore, MD.

[32] Lisha Li, Zhilei Ren, Xiaochen Li, Weiqin Zou, and He Jiang. 2018. How are issue units linked? empirical study on the linking behavior in github. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 386–395.

[33] Zhi-Xing Li, Yue Yu, Gang Yin, Tao Wang, and Huai-Min Wang. 2017. What Are They Talking About? Analyzing Code Reviews in Pull-Based Development Model. *Journal of Computer Science and Technology* 32, 6 (2017), 1060–1075.

[34] Zhiming Liu, Lu Liu, and Hong Li. 2012. Determinants of information retweeting in microblogging. *Internet Research* (2012).

[35] Amy Madden, Ian Ruthven, and David McMenemy. 2013. A Classification Scheme for Content Analyses of YouTube Video Comments. *Journal of Documentation* 69, 5 (2013), 693–714.

[36] Emily E Marsh and Marilyn Domas White. 2003. A Taxonomy of Relationships Between Images and Text. *Journal of Documentation* 59, 6 (2003), 647–672.

[37] Catherine Marshall. 1998. Toward an Ecology of Hypertext Annotation. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*. ACM, New York, NY, 40–49.

[38] Hermann Maurer and Josef Kolbitsch. 2006. Transclusions in an html-based environment. *Journal of Computing and Information Technology* 14, 2 (2006), 161–173.

[39] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. 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 (2019), 1–23.

[40] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[41] Martin Mühlpfordt and Martin Wessner. 2005. Explicit referencing in chat supports collaborative learning. (2005).

[42] M. V. Mäntylä and C. Lassenius. 2009. What Types of Defects Are Really Discovered in Code Reviews? *IEEE Transactions on Software Engineering* 35, 3 (2009), 430–448.

[43] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in Software Engineering Research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, 466–476.
[44] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 25–34.
[45] Theodor Nelson. 1987. *Computer Lib/dream Machine*. Tempus Books of Microsoft Press, Redmond, WA.
[46] Gary M Olson and Judith S Olson. 2000. Distance matters. *Human–computer interaction* 15, 2-3 (2000), 139–178.
[47] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 135 (Nov. 2018), 27 pages.
[48] David Piorkowski, Soya Park, April Yi Wang, Dakuo Wang, Michael Muller, and Felix Portnoy. 2021. How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study. *arXiv preprint arXiv:2101.06098* (2021).
[49] Lars Schneider. 2019. Save time linking resources with autolink references. https://github.blog/2019-10-14-introducing-autolink-references/ Accessed Oct 29, 2020.
[50] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance Factors of Pull Requests in Open-Source Projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (Salamanca, Spain) *(SAC '15)*. Association for Computing Machinery, New York, NY, USA, 1541–1546.
[51] Igor Steinmacher, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Awareness support in distributed software development: A systematic review and mapping of the literature. *Computer Supported Cooperative Work (CSCW)* 22, 2-3 (2013), 113–158.
[52] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 356–366.
[53] Ledyard R Tucker. 1960. Intra-Individual and Inter-Individual Multidimensionality. In *Psychological Scaling: Theory and Applications*, Samuel Messick and Harold Gulliksen (Eds.). Wiley, New York, NY, 155–167.
[54] John Tukey. 1977. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.
[55] Giovanni Viviani, Calahan Janik-Jones, Michalis Famelis, and Gail Murphy. 2018. The structure of software design discussions. In *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 104–107.
[56] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 182 (Nov. 2018), 19 pages. https://doi.org/10.1145/3274451
[57] Matin Yarmand, Dongwook Yoon, Samuel Dodson, Ido Roll, and Sidney S Fels. 2019. "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*. ACM, New York, NY, 489:1–489:12.
[58] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. 2015. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 367–371.
[59] Fiorella Zampetti, Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, and Michele Lanza. 2017. How Developers Document Pull Requests With External References. In *2017 IEEE/ACM 25th International Conference on Program Comprehension*. IEEE, Piscataway, NJ, 23–33.
[60] Yang Zhang, Huaimin Wang, Gang Yin, Tao Wang, and Yue Yu. 2015. Exploring the use of @-mention to assist software development in github. In *Proceedings of the 7th Asia-pacific Symposium on Internetware*. 83–92.
[61] Yan Zhang and Barbara Wildemuth. 2017. *Qualitative analysis of content* (2 ed.). Libraries Unlimited, Santa Barbara, CA, 318–330.
[62] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The Impact of Continuous Integration on Other Software Development Practices: A Large-Scale Empirical Study. In *IEEE/ACM International Conference on Automated Software Engineering*. IEEE, Piscataway, NJ, 60–71.
[63] Sacha Zyto, David Karger, Mark Ackerman, and Sanjoy Mahajan. 2012. Successful Classroom Deployment of a Social Document Annotation System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, 1883–1892.

Received October 2021; revised April 2021; accepted July 2021