Understanding the Impact of GitHub *Suggested Changes* on Recommendations between Developers

Chris Brown
dcbrow10@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Chris Parnin
cjparnin@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

Recommendations between colleagues are effective for encouraging developers to adopt better practices. Research shows these peer interactions are useful for improving *developer behaviors*, or the adoption of activities to help software engineers complete programming tasks. However, in-person recommendations between developers in the workplace are declining. One form of online recommendations between developers are pull requests, which allow users to propose code changes and provide feedback on contributions. GitHub, a popular code hosting platform, recently introduced the *suggested changes* feature, which allows users to recommend improvements for pull requests. To better understand this feature and its impact on recommendations between developers, we report an empirical study of this system, measuring usage, effectiveness, and perception. Our results show that suggested changes support code review activities and significantly impact the timing and communication between developers on pull requests. This work provides insight into the suggested changes feature and implications for improving future systems for automated developer recommendations, such as providing situated, concise, and actionable feedback.

CCS CONCEPTS

• **Human-centered computing** → *Open source software*; • **Software and its engineering** → **Collaboration in software development**.

KEYWORDS

Empirical software engineering, GitHub, suggested changes, online programming communities, developer recommendations

ACM Reference Format:
Chris Brown and Chris Parnin. 2020. Understanding the Impact of GitHub *Suggested Changes* on Recommendations between Developers. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3368089.3409722

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7043-1/20/11...$15.00
https://doi.org/10.1145/3368089.3409722

1 INTRODUCTION

Recommendations between peers is essential for workers to gain knowledge and improve the quality of their work. Boud and colleagues suggest adults predominantly learn from others at work [11]. Software engineering is no exception, as developers frequently learn and share information with each other. For example, studies show software engineers find collaborative development activities, such as *peer code reviews* and *pair programming*, beneficial for learning [3, 17]. Furthermore, research shows that *peer interactions*, or face-to-face recommendations between colleagues during normal work activities, are effective for improving tool adoption [43] and code comprehension [35]. While prior work shows in-person recommendations benefit developers, research also shows these interactions occur infrequently in practice [44]. This is due to a variety of factors, such as the *physical isolation* of remote programmers and distributed development teams [43].

Pull requests (PRs) are the primary method for developers to contribute to projects [26], and consequently, a mechanism through which developers provide recommendations to each other virtually. On GitHub, review comments allow developers to provide feedback on PRs submitted by contributors.[1] Review comments provide ample opportunities for recommendations, such as suggestions for refactoring or conforming to code style guidelines. GitHub recently introduced a new system to facilitate virtual peer interactions called *suggested changes*.[2] This feature allows reviewers to make suggestions to programmers on specific lines of code in pull requests, where developers can easily apply, reject, or edit changes suggested by peers (see Figures 1a-b). After a PR reviewer notices a line of code can be improved, they click on the plus (+) sign on the line of code in question to write a comment and enter their proposed change. Figure 1a shows a reviewer typing their suggested code change for the pull request into the text box. Once the reviewer is finished with their suggestion, clicking on the "Start a review" button submits the suggested change. Finally, the developer who created the pull request sees the suggested change on their code, which is shown in Figure 1b. From here, the developer can commit the change, edit the suggestion, or ignore the proposed modifications. If the developer accepts the change, the suggestion will automatically be integrated into their pull request as a new commit.

We examined GitHub suggestions because it allows us to retroactively examine the impact of a design change for making developer recommendations inside pull requests.[3] The suggested changes

[1] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/commenting-on-a-pull-request#adding-line-comments-to-a-pull-request
[2] https://github.blog/changelog/2018-10-16-suggested-changes/
[3] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/incorporating-feedback-in-your-pull-request

feature is very popular and well-accepted by the GitHub community, with the GitHub Blog noting that developers were "quick to adopt suggested changes" and integrate them into the code review process for projects. Additionally, the blog post reports over 100,000 suggested changes were made within a few weeks of the initial public beta release of the feature, accounting for approximately 4% of pull request review comments and 10% of code reviewers.[4] While suggested changes are increasing in popularity and usage on GitHub, little is known about how developers use them to make recommendations to each other and what impact they have on software development processes. We aim to discover the impact of this feature on developer recommendations and provide implications for increasing the effectiveness of future recommender systems for improving the behavior of software engineers.

To explore the impact of suggested changes on developer recommendations, we seek to answer the following research questions:

RQ1 What types of recommendations do developers make with suggested changes?
RQ2 How effective are recommendation systems on pull requests?
RQ3 What impact do suggested changes have on pull requests?
RQ4 How useful are suggested changes for recommendations between developers?

To answer these questions, we divided our methodology into two phases. The first phase answers the first three research questions by mining repositories to empirically analyze the suggested changes feature on GitHub. The second phase investigates the final research question by collecting qualitative data from developers on their experiences with the system. Our results show this feature effective for recommending a variety of changes to programmers and supports developers in the code review process by helping them make suggestions and decisions on proposed changes quicker. Users also find it useful because for supporting clear communication and easy integration. The main contribution of this work is the first study to empirically analyze the GitHub suggested changes feature.

2 BACKGROUND

2.1 Developer Behavior Recommendations

Research explores making effective recommendations to improve *developer behavior*, or the adoption of useful tools and practices to help software engineers complete development tasks more efficiently. For example, studies show developer behaviors such as using static analysis tools [2] and performing code reviews [39] are useful for improving code quality. Prior work suggests in-person recommendations between programmers are effective for improving developer behavior. For example, research shows that knowledge sharing and learning are benefits of pair programming [17] and peer code reviews [3]. Murphy-Hill explored seven methods software engineers discover new development tools and found that *peer interactions*, or the process of learning about tools from colleagues during work, are the most effective mode of tool discovery [43]. Likewise, research shows peer interactions are effective for improving program comprehension [35] and recommending security tools [62].

[4] https://github.blog/2018-11-01-suggested-changes-update/

Researchers have also explored using systems to make recommendations to improve developer behavior. Robillard and colleagues discuss the importance of Recommender Systems for Software Engineering (RSSEs) for supporting developers and improving decision-making [49]. Additionally, Fischer and colleagues argue that *passive help systems*, which require users to explicitly seek help, are less useful than *active help systems* that can automatically make recommendations to users completing tasks [22]. For example, ToolBox [36], C-3PR [15], and Spyglass [59] are automated systems designed to help programmers discover Unix commands, fix static analysis bugs, and search code more efficiently.

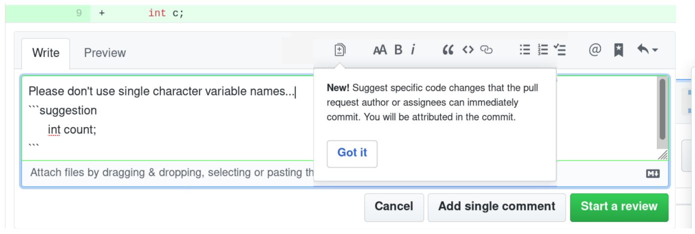
Finally, prior work shows integrating recommender systems in the code review process is effective for improving developer behavior. Singh and colleagues suggest static analysis tools can support developers during code reviews [51]. Balachandran discovered Review Bot, a system that automates static analysis and reviewer recommendations, improved code quality and reduced reviewer effort at VMWare [4]. Furthermore, researchers at Facebook found running development tools at *diff time*, or on code submitted to be reviewed before merging, increased the fix rate of reported bugs approximately 70% [19]. Suggested changes incorporate recommendations into the code review process, and this research aims to discover how this feature influences developer behavior.

2.2 Making Recommendations on GitHub

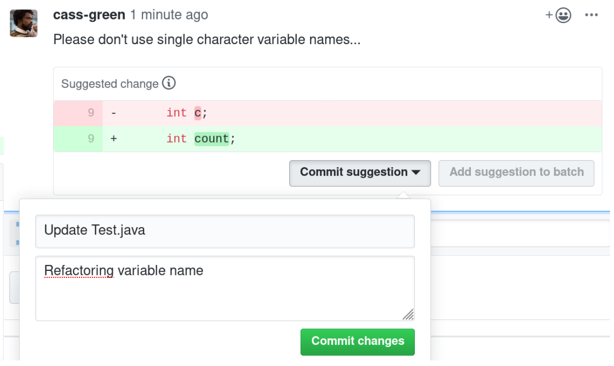
The GitHub platform has a variety of methods for developers to make recommendations to each other. For example, pull requests allow developers to "propose and collaborate on changes to a repository."[5] There were over 87 million pull requests merged to GitHub projects in 2019 [23], and research shows this system is effective for making suggestions to improve repositories [24]. Similarly, GitHub issues are useful for tracking a variety of information for repositories, with over 20 million closed last year [23].[6] Bissyandé and colleagues suggest that, while the majority of issues are reported "bugs", those labeled as "feature" or "enhancement" are "equally important for issue reporters" [7, p. 193]. Furthermore, GitHub has explored automated recommendations to developers, such as generating alerts for potential security vulnerabilities[7] and automating security fixes.[8] To analyze the impact of recommendations on pull requests, we compare suggested changes with another system for suggestions between developers: *pull request review comments*.

Review comments differ from general pull request discussion and issue comments because they are located on specific lines of code during pull request reviews. Furthermore, reviewers can also suggest code changes to developers in pull request review comments by using code fences to start and end the proposed block of code. For example, Figure 2 presents the same code recommendation as the suggested change in Figure 1b using a review comment. Suggested changes are instances of pull request review comments for developers to propose improvements to users on pull requests during code reviews, however they also provide a user interface

[5] https://help.github.com/en/articles/creating-a-pull-request
[6] https://help.github.com/en/articles/about-issues
[7] https://help.github.com/en/github/managing-security-vulnerabilities/about-security-alerts-for-vulnerable-dependencies
[8] https://help.github.com/en/github/managing-security-vulnerabilities/configuring-automated-security-updates


Write
Preview
Please don't use single character variable names...
```suggestion
int count;
```
Attach files by dragging & dropping, selecting or pasting them.
New! Suggest specific code changes that the pull request author or assignees can immediately commit. You will be attributed in the commit.
Got it
Cancel
Add single comment
Start a review

(a) Reviewer adds comment and suggested change to modified line of code



(b) Developer can apply suggested change and commit to PR

Figure 1: GitHub Suggested Changes example

that allows users to automatically apply recommendations and commit suggestions to the PR code. In this work, we compare both pull request recommendation systems to analyze how they impact recommendations between peers and improve specific portions of code contributions from developers on open source repositories.

3 METHODOLOGY

To explore the impact of the GitHub suggested changes feature on recommendations between developers, we divided our evaluation into two phases for gathering data to answer our research questions.

3.1 Phase 1: An Empirical Study on GitHub Suggested Changes

The first phase of this work mines public repositories to explore the usage and effectiveness of suggested changes on GitHub.

3.1.1 Data Collection. To automatically detect suggested changes on pull requests, we created a script to programmatically search for instances of the md"'suggestion tag in pull request review comments (See Figure 1a). This indicates that a suggestion was proposed by a reviewer on a pull request using this system. We used the PyGithub API[9] to sort projects by most recently updated pull requests and parse comments in order to collect current uses of the suggested changes feature. Then we compiled a list of pull request comments with suggested changes to characterize the types of code changes developers recommend with this system.

To examine the effectiveness of suggested changes and their impact on pull requests, we analyzed the top-forked repositories on GitHub that contain pull requests with comments using suggested changes to collect instances of suggestions and pull request review comments with code. PyGithub provides an API to retrieve pull request review comments and, similar to our suggested changes detection technique, we searched for instances of the code fence to determine if comments contain code snippets. To limit our search, we only observed activity on pull requests after October 2018, which is when the suggested changes feature was introduced as a public beta release on GitHub. Overall, we analyzed a total of 152,030 pull request review comments and found 17,712 suggested changes on 51,250 pull requests. The list of projects and activity collected for this evaluation are presented in Table 1. The script used to automatically detect suggested changes and our resulting dataset are publicly available online.[10]

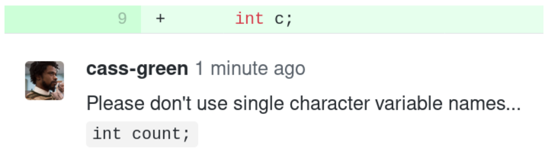


Figure 2: Pull Request Review Comment example

3.1.2 Categorizing Suggested Changes. To characterize types of changes developers suggest for RQ1, we randomly sampled 100 recently updated pull requests with an instance of the md"'suggestion tag in the comments. Two researchers performed an *open* coding analyzing review comments and code modifications recommended in suggested changes to identify types of changes recommended by developers with this feature (inter-rater agreement = 71%, Cohen's κ = 0.5942). Then, the raters came together to discuss derived categories and come to an agreement. The main source of discrepancies arose from determining the functionality of suggested changes and deciding if suggestions were correcting or improving lines of code.

3.1.3 Determining Recommendation Effectiveness. To answer RQ2, we evaluated effectiveness for recommendations between developers with suggested changes and pull request review comments on GitHub by analyzing *acceptance* and *timing*. We further divide these criteria into subcategories: *contribution acceptance* and *recommendation acceptance*, and *contribution time*, *recommendation time*, and *recommendation acceptance time*.

Acceptance. This criteria examines the impact of users incorporating proposed changes from another developer. *Contribution acceptance* refers to how frequently contributions from developers are merged into projects. In the pull-based software development

[9] https://pygithub.readthedocs.io/en/latest/

[10] https://github.com/chbrown13/suggestions

Table 1: *Suggested Changes* Study Data

Project	Primary Lang.	Suggested Changes	Comments (w/ Code)	PRs
qmk/qmk_firmware	C	8525	9127 (675)	3600
nodejs/node	JavaScript	2252	14364 (818)	5889
rust-lang/rust	Rust	2615	24139 (1068)	8475
go-gitea/gitea	Go	1317	6604 (338)	2822
rapid7/metasploit-framework	Ruby	754	4057 (517)	1252
Qiskit/qiskit-terra	Python	626	3505 (204)	1735
kubernetes/kuberbetes	Go	556	52297 (2645)	12106
qgis/QGIS	C++	481	2893 (37)	2548
neovim/neovim	Vim script	208	3087 (163)	1746
python-pillow/Pillow	Python	156	354 (19)	650
mono/mono	C#	134	6043 (182)	5821
lydiahallie/javascript-questions	Markdown	49	43 (2)	231
jpmorganchase/quorum	Go	10	192 (13)	193
firebase/quickstart-android	Java	5	89 (6)	155
mavlink/qgroundcontrol	C++	5	261 (14)	863
qbittorrent/qBittorrent	C++	5	3966 (205)	486
ironhack-labs/lab-advance-querying-mongo	Markdown	4	159 (12)	753
kenwoodjw/python_interview_question	Markdown	3	2 (0)	38
lballabio/QuantLib	C++	3	57 (4)	146
Azure/azure-quickstart-templates	PowerShell	2	2718 (2)	1630
h5bp/Front-end-Developer-Interview-Questions	HTML	1	81 (2)	56
qunitjs/qunit	JavaScript	1	77 (11)	55
Total:	N/A	17712	134318 (6937)	51250

model, merged pull requests indicate that a contribution from an external developer was integrated into the main code base for a repository [24]. Research suggests contribution acceptance is a useful metric for measuring the success of GitHub projects [38] and is vital for the maintenance and evolution of open source software [40]. We use this to determine if the existence of recommendation comments impacts the outcome of pull requests. To analyze the impact of the recommendations on contribution acceptance, we compare the percentage of merged pull requests containing a code suggestion (suggested change or review comment with fenced code) compared to those without code recommendations.

Recommendation acceptance. This refers to how often the recommendations are approved by developers. To determine how effective developers find code suggestion systems, we measured the percentage of suggested changes and fenced code review comments incorporated into pull requests from our dataset. For suggested changes, we programmatically analyzed pull request comments by creating a script to detect suggested changes and extract code between md"'suggestion and the ending md"' to determine the status of suggested changes since this feature is currently not supported by the GitHub API.[11] Then, we checked whether the extracted code existed further changes to the file by analyzing subsequent commits to the pull request. If so, we consider the suggestion accepted by the developer. Similarly, we used this process to determine acceptance for pull request review comments by examining code inside fence tags in future commits on the pull request.

[11] https://github.community/t5/GitHub-API-Development-and/Accessing-the-new-quot-GitHub-Suggestions-quot-via-API-public/td-p/13922

Time. This criteria explores the impact of pull request recommendations on overall development time. *Contribution time* is defined by the amount of time to accept contributions from developers. Yu and colleagues analyzed characteristics that influence pull request evaluation latency, or the amount of time to review pull requests on GitHub [63]. We aim to determine if the presence of code suggestions impacts the latency of contributions from developers. To measure contribution time, we used the PyGithub API to calculate the difference between the pull request creation time and when the time it is merged into the repository by a project maintainer based on whether the review comments contain a code suggestion or not.

Recommendation time. Recommendation time examines the amount of time it takes developers to make recommendations with a given system. Research shows that minimizing development time is important because the longer a defect exists in code the more expensive and difficult it becomes to fix [32]. To determine the impact of code suggestions on time, we compared how quickly developers make recommendations with suggested changes and code in pull request review comments. To measure impact of time on reviewers, we calculated the *recommendation time* as the amount of time between pull request creation and for a developer to add a code recommendation comment with the md"'suggestion or md"' tags.

Similarly, *recommendation acceptance time* refers to the amount of time it takes developers to decide on recommendations on pull requests. For code suggestions, we measured the acceptance time as the amount of time between a reviewer commenting on a pull request using a suggested change or pull request review comment with fenced code until the time a subsequent commit adding the recommended line of code was added to the pull request.

3.1.4 *Evaluating Pull Request Impact.*

For RQ3, we were specifically interested in the impact of suggested changes on pull requests. To study this, we calculated a variety of pull request metrics used in existing software engineering literature.

Pull Request Characteristics. To further analyze the impact of suggested changes on development processes, we used metrics introduced by Gousios and colleagues for the *pullReqs* dataset exploring the impact of pull requests on projects [25]. Their dataset analyzed approximately 350,000 pull requests on 900 GitHub repositories, and also provides a suite of characteristics to describe GitHub pull requests and quantify their impact on code repositories. To discover the impact of suggested changes on pull requests, we used the PyGithub API to collect metrics for the following characteristics to analyze pull requests:

- ***lifetime_minutes***: Number of minutes between pull request opening and closing (if closed)
- ***mergetime_minutes***: Number of minutes between pull request opening and merging (if merged)
- ***num_commits***: Total number of commits for a pull request
- ***src_churn:*** Total number of lines changed
- ***files_changed:*** Total number of files touched by pull request
- ***num_commit_comments:*** Number of review comments
- ***num_issue_comments:*** Number of discussion comments
- ***num_participants:*** Number of discussion participants

We selected these characteristics from the original *pullReqs* dataset because they describe development activity and feedback during the code inspection process. We were primarily interested in these metrics to help better understand how suggested changes influence contributor and reviewer behavior on pull requests. To determine the impact of suggested changes, we compare these characteristics for pull requests with suggested changes and pull requests without suggested changes.

3.2 Phase 2: Developer Feedback on Suggested Changes

The second phase of this evaluation consists of a survey to analyze developer experiences and perceptions on the usefulness of the suggested changes feature.

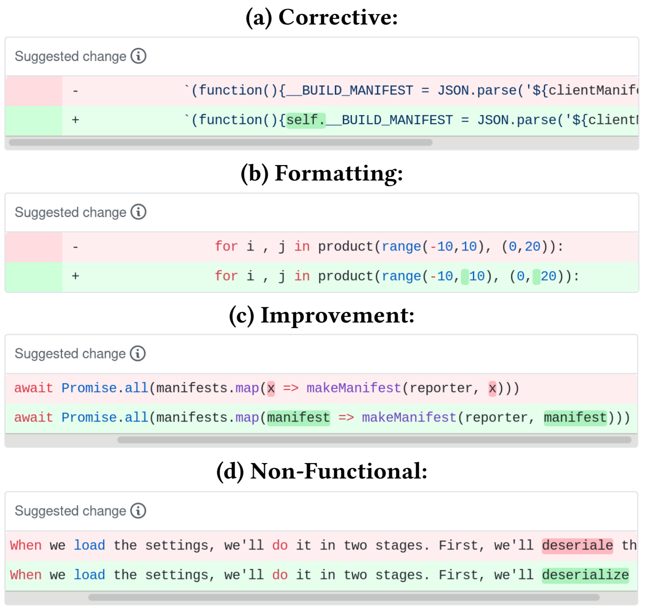
3.2.1 *Data Collection.*

To answer RQ4, we surveyed developers who interacted with suggested changes on GitHub. Surveys were emailed to users with publicly available email addresses who either received a suggested change on their pull request or made a suggested change on another developer's pull request within the last six months. Our survey asked users how useful they found this feature using a 5-point Likert scale. We also included free response questions for participants to provide additional feedback on what specifically they find useful or unuseful about the system as well as how they integrate this feature into their projects.

3.2.2 *Determining the Usefulness of Suggestions.*

We emailed surveys to a total of 580 GitHub users who interacted with suggested changes and received 43 responses (7.4% response rate). Throughout the remainder of this paper, we use the C- prefix to describe a *suggestee*, or a contributor who received a suggested change on their pull request, and the R- prefix to indicate a *suggester*, or a reviewer who made a comment with the suggested changes feature on a pull request. We aggregated Likert scores to quantitatively examine usefulness, then two experts *open* coded the open-ended responses from developers on the useful aspects of suggested changes (72%, Cohen's $\kappa = 0.6828$). The researchers discussed derived categories and came to an agreement on themes found in developer comments.

4 RESULTS


(a) Corrective:

Suggested change

```
-            `(function(){__BUILD_MANIFEST = JSON.parse('${clientManif
+            `(function(){self.__BUILD_MANIFEST = JSON.parse('${clientI
```

(b) Formatting:

Suggested change

```
-                for i , j in product(range(-10,10), (0,20)):
+                for i , j in product(range(-10, 10), (0, 20)):
```

(c) Improvement:

Suggested change

```
await Promise.all(manifests.map(x => makeManifest(reporter, x)))
await Promise.all(manifests.map(manifest => makeManifest(reporter, manifest)))
```

(d) Non-Functional:

Suggested change

```
When we load the settings, we'll do it in two stages. First, we'll deseriale th
When we load the settings, we'll do it in two stages. First, we'll deserialize
```

Figure 3: Categories of Suggested Changes

4.1 RQ1: Types

4.1.1 *Categories.*

We derived four categories to describe types of suggested changes. Each of the categories identified is presented with examples below:

Corrective: The corrective category refers to using suggested changes to fix issues found in code. Bacchelli and colleagues found that fixing defects is the primary motivation for code reviews reported by developers at Microsoft [3]. Additionally, Chillarege and colleagues found that *algorithm errors* are the most common type of defects discovered during code inspections [1]. Algorithm errors include those that deal with correctness or bugs in code implementations. In Figure 3a, we observed an instance of a corrective suggested change by a developer on a pull request. The contributor referred to a variable as a global variable instead of a class variable, and the reviewer proposes a fix by adding the `self` keyword.[12]

Formatting: This category represents code changes that solely impact the presentation of the code without changing the functionality. This also includes refactoring, or the process of restructuring code without changing its behavior, which research shows is beneficial for improving code quality [55]. Bacchelli and colleagues also reported developers find code reviews useful for ensuring code

[12] https://github.com/zeit/next.js/pull/7696#discussion_r302333269

styles and standards are consistent for development teams [3]. Finally, Mäntylä et al.'s evolvability defects contain the subcategory of *visual representation*, which refers to the formatting of code [37]. An example formatting suggestion is presented in Figure 3b.[13] In this case, the reviewer suggests improvements to the spacing of the code to adhere to Python Enhancement Proposal (PEP 8) coding style guide and whitespace standards [47].

Improvement: Improvement suggested changes occur when developers recommend a code change to refactor or optimize a user's code. Greiler notes that developers at Microsoft reported that improvements to code quality is the most important benefit of code reviews.[14] Chillarege finds that even correct algorithms may have additional concerns, such as the implementation's efficiency, that may need improvement during reviews [1]. Similarly, Mäntylä's evolvability defects include problems in the structure and organization of the code that can be optimized [37]. Figure 3c presents an example of an improvement suggested change, where the reviewer recommends an improvement to the readability of the code by renaming a variable from `x` to `manifest`.[15]

Non-Functional: This category represents changes developers suggest that don't impact code, including suggestions to reword and fix spelling and grammar problems in documentation and code comments. Non-functional documentation issues are prevalent in software, for example Beller et al. found that documentation changes are the most frequent type of fix applied for code reviews in open source software [6]. Similarly, Mäntylä and colleagues found that evolvability defects, or problems with the understandability and maintenance of software, including documentation errors, are the most common type of defects found during code reviews [37]. An example of a non-functional defect is presented in Figure 3d. In this case, the reviewer discovers a typo misspelling *deserialize* and makes a suggestion to fix the error.[16]

4.1.2 Usage. Out of 100 randomly sampled uses of the suggested changes feature, we found the most popular suggestions were non-functional changes. Table 2 presents our results for each category. While non-functional changes were the most frequent type of change suggested, we also found that this feature is often used for general code improvements to optimize code. This shows that suggested changes are an effective system because of their ability to recommend a wide variety of code changes to developers during reviews, specifically for recommendations for non-functional and code improvement changes to contributions from developers.

Table 2: Suggested Change Categories

	n	Percentage
Non-Functional	36	36%
Improvement	34	34%
Corrective	16	16%
Formatting	14	14%

[13] https://github.com/numba/numba/pull/4204#discussion_r310598073
[14] https://www.michaelagreiler.com/code-reviews-at-microsoft
[15] https://github.com/gatsbyjs/gatsby/pull/13471#discussion_r277948539
[16] https://github.com/microsoft/terminal/pull/1258#discussion_r293932790

4.2 RQ2: Effectiveness

To determine the effectiveness of recommendation systems on pull requests, we evaluated their impact on acceptance and timing.

4.2.1 Acceptance.

Contribution Acceptance. In total, 69% of pull requests analyzed in our dataset (n = 35,521) were merged. We analyzed the merge rate for pull requests with and without code suggestions, presented in Table 3. Our results suggest that both groups have approximately the same merge rate, and using a chi-squared test we found no significant difference in the existence of code suggestions (suggested changes or review comments with code) on the outcome of pull requests ($\chi^2 = 0.0182$, $p = 0.8928$, $\alpha = .05$). These results show that, while suggestions are useful for helping developers resolve problems in code reviews, they do not have a major influence on whether contributions are accepted and merged into repositories.

Recommendation Acceptance. We also observed acceptance for suggested changes and review comments with fenced code among GitHub users. Overall, we observed 10,556 out of 17,712 (60%) suggested changes and 65 out of 6,937 (1%) pull request review comments with markdown code were accepted by developers. Table 4 presents our results for acceptance for each system. Using a chi-square test to analyze the outcome of recommendations for each group, we found that the difference in the acceptance of suggested changes and pull request review comments was statistically significant ($\chi^2 = 6961.3765$, $p < 0.00001$, $\alpha = .05$). While code suggestions do not significantly increase, nor decrease the overall acceptance of contributions from developers, suggested changes facilitate refinements to those contributions when compared to other recommendation methods, such as review comments.

Table 3: Contribution Acceptance Rate

Pull Requests	*n*	Rate
with suggestions	6982	69.4%
without suggestions	44268	69.3%

Table 4: Recommendation Acceptance Rate

Type	*n*	Rate
suggested changes	17712	59.6%
review comments with code	6937	0.9%

4.2.2 Timing.

Contribution Time. To measure the impact of pull request code suggestions on time, we examined the amount of time to accept code contributions for PRs with suggested changes or review comments with fenced code compared to those with neither system. The results of this analysis are presented in Table 5. On average, we discovered that pull requests with suggestions take over twice as long to merge into repositories. Using the Mann-Whitney-Wilcoxon test, we found that pull requests with code suggestions take significantly longer to be accepted than non-suggestion pull requests ($W = 87857043$, $p < 0.00001$, $\alpha = .05$). Contributions with recommendations may take longer to be accepted due to increased discussions and comments from reviewers to fix issues in pull requests.

Recommendation Time. We analyzed the amount of time for reviewers to comment with recommendations to determine their impact on overall development time. Recommendation time refers to the amount of time for reviewers to comment with suggestions. Our results, displayed in Table 6, found a significant difference between suggested changes and pull request review comments with code using the Mann-Whitney-Wilcoxon test (W = 49186174, $p <$ 0.00001, α = .05). This indicates that reviewers are able to make recommendations significantly faster using the suggested changes feature compared to code within review comments, and this system allows them to efficiently provide feedback to developers on pull requests during the code review process.

Recommendation Acceptance Time. We also found that suggested changes have a major impact on the amount of time to accept recommendations. Table 7 presents the average acceptance time for developers for each type of recommendation. We found a significant difference in the amount of time taken to accept a recommendation for each system (W = 256013, p = 0.0001, α = .05). This shows that developers are able to make decisions to accept recommendations significantly faster with suggested changes than with only markdown code in pull request review comments. Overall, our timing results show that, although contributions take longer overall to get merged, suggested changes facilitate recommendations on pull requests better by decreasing the amount of time needed to provide feedback and accept recommendations during reviews.

Table 5: Contribution Time (in days)

Type	*n*	Average	Median
with suggestions	6982	16.4	5.0
without suggestions	44268	6.4	1.1

Table 6: Recommendation Time (in days)

Type	*n*	Average	Median
suggested changes	17712	10.5	0.7
review comments with code	6937	14.6	1.9

Table 7: Recommendation Acceptance Time (in days)

Type	*n*	Average	Median
suggested changes	17712	5.4	0.3
review comments with code	6937	8.0	0.7

4.3 RQ3: Pull Request Impact

We were also interested in the impact of the suggested changes feature on GitHub pull requests. Our dataset included 4,319 PRs with suggested changes and 46,931 without them.

4.3.1 Pull Request Characteristics. To further understand the impact of suggested changes on pull requests, we measured existing metrics used to examine PRs on GitHub [25]. These results are presented in Table 8. We used the Mann-Whitney-Wilcoxon test (α = .05) to compare pull request characteristics for PRs containing suggested changes to those that do not. We discovered that pull requests with suggested changes take significantly longer to be closed (***lifetime_minutes***) and accepted (***mergetime_minutes***). This may be due to the fact that pull requests with suggested changes have more complex development activity. For example, pull requests utilizing suggested changes have significantly more commits (***num_commits***) and modified lines of code (***src_churn***) in contributions from developers. This is understandable, because accepted suggested changes automatically apply additional commits to pull requests. While this feature helps resolve problems during reviews, it also adds more time to code inspection processes.

Additionally, we found that suggested changes significantly impact feedback during code reviews. Our results show that pull requests using the suggested changes feature have significantly more review comments within the code (***num_commit_comments***), general discussion about contributions (***num_issue_comments***), and developers contributing to discussions about pull requests (***num_participants***). This shows suggested changes are useful for providing feedback to developers, facilitating communication, and discussing improvements to pull requests.

4.4 RQ4: Usefulness

4.4.1 Likert Scale. Figure 4 presents the Likert scale question results on how useful the survey respondents found the suggested changes feature. Of the 43 survey responses we received from GitHub users, 24 responses from were from suggestees and 19 responses were from suggesters. Approximately 92% of suggestees (n = 22) and 79% of suggesters (n = 15) who participated in the survey responded that the suggested changes feature is Useful or Very Useful. Furthermore, no GitHub users who interacted with this feature and completed our survey reported that it was Not at All Useful. The one suggester who ranked the system as Somewhat Useful complained about the suggested changes functionality that incorporates suggestions as separate pull request commits and desired a “force push” option to squash the commits and better adhere to their team’s development workflow. Our survey results suggest GitHub users find suggested changes to be an effective system for both receiving recommendations from peers as well as making recommendations to developers on GitHub.

4.4.2 Qualitative Feedback. To further examine this feature, we asked participants to provide open-ended responses describing what they find useful about suggested changes. The primary reasons developers found them useful is that they are effective for communication, provide concise information, and are presented at convenient times to developers. We expand on these findings from our open-ended feedback in the discussion when presenting design principles for improving recommendation systems for software engineering. Here, we define and provide examples of each category derived from our qualitative coding to summarize developer responses on what they found useful about suggested changes.

Table 8: Pull Request Characteristics

Characteristic	Dataset	Mean	Median	*p-value*
lifetime_minutes***	with suggested changes	34799.26	8190.43	-
	without suggested changes	18993.48	2489.12	$p <$ **0.00001**
mergetime_minutes***	with suggested changes	23645.15	5900.27	-
	without suggested changes	10378.82	1776.32	$p <$ **0.00001**
num_commits***	with suggested changes	8.62	4	-
	without suggested changes	6.72	1	$p <$ **0.00001**
src_churn***	with suggested changes	866.86	133	-
	without suggested changes	3212.64	26	$p <$ **0.00001**
files_changed	with suggested changes	7.71	2	-
	without suggested changes	11.54	2	$p = 0.5051$
num_commit_comments***	with suggested changes	13.56	7	-
	without suggested changes	2.00	0	$p <$ **0.00001**
num_issue_comments***	with suggested changes	8.63	5	-
	without suggested changes	5.03	3	$p <$ **0.00001**
num_participants***	with suggested changes	3.18	3	-
	without suggested changes	2.30	2	$p <$ **0.00001**

*** denotes statistically significant results (**p-value < 0.05**)

Actionability: One reason developers found suggested changes useful was because of the actionability of the recommendations. This refers to the ease with which users can act on recommendations. With suggested changes, developers can immediately commit proposed code changes to their on pull requests. Survey participants frequently mentioned suggested changes are useful because they can be automatically applied to their code.

> *"the fact that small changes can be applied immediately, and the fact that they can be described by the reviewer in a way that a button fixes it instead of going to your code"* (C3)

Communication: We also discovered that suggested changes are effective for communication in our survey feedback. This theme refers to the understanding and clarity in transferring knowledge between peers through the recommendation system. Participants reported that suggested changes provide clear communication between developers and reviewers during the code review process.

> *"We can understand reviewer's intention more. If not using this feature, there are only pull request comment text, so we may misunderstand reviewer's intention"* (C16)

Code: Another reason participants found the suggested changes feature useful is because reviewers can propose changes as code. Several participants mentioned they preferred the ability to give and receive recommendations as code rather than writing out suggestions as text in comments.

> *"It is very convenient that the reviewer can write what they suggest to change in code instead of formulating it in words (which will often be longer)"* (R6)

Conciseness: This category refers to the brevity of recommendations with suggested changes. Many developers found the concise code recommendations to be useful when using this system. Examples of this can found in responses such as the one below where a suggester found suggested changes useful because they can:

> *"Suggest small one line changes directly and concisely"* (R9)

Ease of Use: In our feedback from developers, many participants stated that suggested changes are effective because they are easy for suggesters and suggestees to use. GitHub users commended this feature for its intuitive interface and ability to effortlessly integrate into the code review process for pull requests.

> *"It's really easy to use, it's really easy to accept the suggestion so some works are really easy"* (C15)

Location: The location category refers to where recommendations are made to users. We found that developers found the suggested changes feature useful because it makes recommendations directly on the line of code to be improved. For example, one participant noted that with this feature there is:

> *"No need to leave the pull request page to make a suggested change, [and it's] easy to see what is being suggested"* (R12)

Scalability: Another benefit of suggested changes reported by developers is the scalability of this feature. This refers to the ability to allow users to easily make and receive numerous recommendations. GitHub allows users to suggest multiple changes on pull requests with this feature, which respondents reported to be useful.

> *"It...gives me the ability to suggest multiple options during review"* (R11)

Timing: Developers also noted the timing of suggested changes makes them useful. This category refers to the speed to which users can make and address recommendations with this feature. Many survey respondents commented on how effectively suggested changes impact the time spent doing code reviews.

> *"Being allowed to add specific changes speeds up the review process. Sometimes it is easier to make the changes yourself rather than make a suggestion and wait for a change"* (C17)

No Response: In some cases, participants did not respond on the usefulness of suggested changes. This question was optional in the survey and we cannot conclude the reason developers chose not to respond is due to not finding anything useful about the feature.

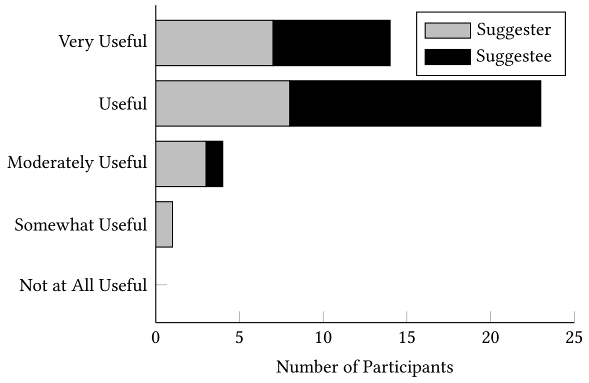


Figure 4: Survey Responses on the Usefulness of *Suggested Changes*

5 DISCUSSION

Our results show that GitHub suggested changes are useful for making a variety of recommendations to code contributors and reviewers. This feature impacts the amount of time to make and accept suggestions during code reviews, facilitates recommendations on pull requests, and improves development activity and communication between developers in the pull-based development model. Based on these findings, we propose incorporating *user-driven communication* and *workflow integration* to improve automated systems for recommending developer behaviors.

5.1 User-Driven Communication

This principle refers to how information is conveyed to users. We found one of the most useful aspects of suggested changes is their ability to effectively communicate to developers. Prior work suggests poor communication prevents static analysis tool adoption [30] and frustrates developers during interactions with bots [61]. To improve automated recommendations, we present two themes derived from our results to implement user-driven communication: *communication* and *conciseness.*

5.1.1 Communication. Participants found suggested changes effective for transferring knowledge and providing clear communication between peers. For instance, users replied this system is useful because it "*lets someone else directly make changes instead of writing out instructions on how to make changes*" (C10), "*gives the suggestion in a very clear way*" (R5), and provides "*easy information on what to change in your pull request*" (C5). We found suggested changes improved communication on pull requests by allowing more feedback in comments and more developers to participate in discussions. In prior work, Cerezo and colleagues note the importance of clear communication to developers by proposing implementing *user-driven conversations* as opposed to *bot-driven* techniques for improving recommender chatbots [16]. We propose implementing clear communication focused on users to improve automated developer behavior recommendations. For example, using specific and clear language to present benefits and educate developers on utilizing security tools can help overcome barriers to adoption [62].

5.1.2 Conciseness. We also found users appreciated the brief and compact nature of recommendations with suggested changes. R11 mentioned they liked "*it uses less words.*" Many developers also preferred recommendations as code. For example, users noted this feature "*can quickly and precisely show what change they expect. Describing the change with words is pretty annoying*" (C12), "*removes guesswork from interpreting a prose explanation*" (R4), and "*removes all ambiguity about what I'm asking for if I can just directly put the code there*" (R12). Prior work also suggests conciseness is important. For example, Wasserman's guidelines for designing interactive software systems states "a much shorter message will frequently suffice" [60, p. 388]. Blatt and colleagues also report software engineers desire concise emails, and often prefer instant messaging systems because they are shorter and quicker than emails [9]. To increase developer behavior adoption, we suggest recommendations incorporate concise messages to potential users. For instance, Johnson and colleagues propose using concise messages in a *quick fix design* to automatically provide fixes to reported bugs and encourage developers to adopt code-checking systems [30].

5.2 Workflow Integration

This implication emphasizes the importance of integrating recommendations effectively into developer workflows. Survey participants found suggested changes useful because they easily integrate into development processes. Furthermore, studies show integrating into developer workflows is vital for the adoption of static analysis tools [30], automated recommendations [12], development tools at large software companies [21], and program repair bots [57]. Here, we present three categories developers reported made suggested changes useful: *timing*, *location*, and *actionability.*

5.2.1 Timing. Many survey respondents commented on how suggested changes impacted time during reviews. For example, participants stated this feature "*lets me do reviews much faster*" (R3), "*accelerates getting pull requests accepted*" (C4), "*it's great to be able to quickly apply changes*" (C23), and "*it's often quicker both to suggest a minor change*" (R8). We also found that, while suggested changes lengthen the overall pull request review process, they help developers make and decide on recommendations during reviews significantly faster. Previous research also shows the importance of timing in developer recommendations. For example, Viriyakattiyaporn and colleagues found the inability to deliver suggestions in a timely manner discouraged programmers from adopting code navigation recommendations from Spyglass [58]. We suggest making convenient and timely recommendations within the workflow of developers to increase adoption of useful behaviors. For example, researchers at Facebook found presenting static analysis tool output at *diff time* encouraged developers to fix more bugs compared to times outside of developers' workflows, such as in overnight builds, which had a nearly 0% fix rate [19].

5.2.2 Location. Our results suggest that location, or the placement of notifications, are also important for developer recommendations. For example, C24 praised the location of suggested changes saying that there's "*no need to leave the pull request page to make a suggested change.*" C22 also added this feature is useful because it "*shows suggested code changes integrated with the actual source.*" Software engineering research also shows the importance of recommendation location. For example, Smith and colleagues found that *in situ* code navigation tools increased efficiency and was preferred by developers [52]. Brown and colleagues also discovered developers prefer tool recommendations situated in their projects, as opposed to from external systems such as email [13]. We recommend placing suggestions in advantageous locations for developers without interrupting their workflows. For example, placing recommendations within IDEs can minimize *visual momentum* and help programmers feel less disoriented in their development environment [18].

5.2.3 Actionability. Actionability refers to the ease with which users can act on recommendations. Participants noted the suggested changes feature is useful because it allows developers to immediately apply suggestions and commit proposed changes to their code. For example, C8 stated they appreciated "*the ability for people to be able to suggest changes and to be able to incorporate those changes immediately*" and C9 mentioned they liked recommendations can be "*accepted right away, without requiring copy pasting and committing on my side.*" Prior work by Heckman and colleagues found actionable alert identification techniques (AAITs) in static analysis tools help identify defects in code earlier [28]. Based on our findings, we advise integrating actionability into automated recommendations to allow users to conveniently adopt tools and practices. One example of this is automating developer behaviors, such as prior work by Evans et al. which shows that automatically enabling security checks prevents security vulnerabilities better than requiring users to explicitly turn them on [20].

6 LIMITATIONS AND FUTURE WORK

An external threat to this work is that we only examined public open source repositories on GitHub. This may not generalize to closed source software projects or repositories hosted on other code hosting platforms. To mitigate this threat, we used a diverse set of GitHub repositories varying in organization, size, language, etc. For RQ1, we used a random sample to avoid bias from the same users and projects in our data. For our RQ2 and RQ3 dataset, we analyzed the top-forked repos to have more opportunities to collect suggested changes and review comments with fenced code since both require pull requests and GitHub recommends forking projects to create pull requests.[17] Another limitation is we analyzed PR acceptance using the GitHub Merged status, however research suggests pull requests that appear as non-merged may actually be merged [31]. To overcome this, we observed additional metrics to study the impact of suggested changes on pull requests. We acknowledge there may be limitations to our technique for analyzing suggested changes and review comments with fenced code, however neither feature is supported by the GitHub API. Additionally, there are other methods for recommendations between developers on GitHub, such as discussion comments. In this work, we analyzed code suggestions on pull requests to explore the impact of suggested changes on facilitating recommendations between peers.

In future work, we aim to explore implementing an automated recommendation system leveraging suggested changes. We envision this system suggesting bug fixes to programming errors based on the output of code analysis with suggested changes while recommending static analysis tools to developers. Additionally, we hope to evaluate the impact of other novel systems, such as GitHub Actions[18] or GitLab AutoDevOps,[19] on developer recommendations and behavior. Furthermore, while this work focuses on code reviews, we are interested in discovering how integrating systems into other software development processes, such as pair programming, continuous integration, debugging, and testing, can impact recommendations to software engineers and developer behaviors.

7 RELATED WORK

This research builds on prior work related to peer learning and developer recommendations among software engineers in other online programming communities. For example, previous studies have examined various methods for software developers to learn and share knowledge through platforms such as Stack Overflow [45],[20] Twitter [50],[21] Hacker News [5],[22] and Wikipedia [48]. Furthermore, researchers and toolsmiths have proposed and examined virtual recommendations between software developers through processes such as live-coding [8], continuous social screencasting [42], gamification [53], networked workstations [36], logs of user actions [34], and crowdsourced socio-technical content on social media [54]. Our work aims to discover the impact of suggested changes on recommendations between developers.

Additionally, prior software engineering research has empirically explored GitHub functionality. Researchers have explored GitHub features such as pull requests [26], issues [7], issue tracker labels [14], issue and pull request links [33], project forks [29], commit comments [27], README files [46], stars [10], and badges [56] for influencing developer behavior and software engineering practices. Software engineering researchers have also explored using GitHub processes to recommend developer behaviors. For example, Mirhosseini et al. used automated pull requests to encourage developers to upgrade out-of-date software dependencies [41]. We add to this work by analyzing a novel GitHub feature, *suggested changes*, and exploring the impact of this system on pull requests, code reviews, and recommendations between developers.

8 CONCLUSION

In this paper, we empirically evaluated the new suggested changes feature on GitHub to discover its impact on recommendations between developers. We found that this system significantly impacts the time, development activity, and review process of pull requests and is effective for supporting reviewers in providing feedback and developers in accepting recommendations quickly. Feedback from developers provides insight into improving automated recommendations to software engineers by incorporating user-driven communication and smooth workflow integration. As opportunities for face-to-face recommendations between developers decline, we propose incorporating these into automated systems to improve adoption of developer behaviors.

[17] https://help.github.com/en/articles/fork-a-repo

[18] https://github.com/features/actions

[19] https://docs.gitlab.com/ee/topics/autodevops/index.html#auto-code-quality-starter

[20] https://stackoverflow.com/

[21] https://twitter.com

[22] https://news.ycombinator.com

ACKNOWLEDGEMENTS

This work was funded by NSF Grant #1714538. We thank all of the developers who participated and contributed to this work. Additionally, we would like to thank Sarah Elder and Zhe Yu for their efforts in the qualitative coding processes.

REFERENCES

[1] Gustavo Alonso and Claus Hagen. 2000. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering* 18, 10 (oct 2000), 943–958. https://doi.org/10.1109/32.879818

[2] Nathaniel Ayewah and William Pugh. 2010. The Google FindBugs Fixit. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA 2010)*. Association for Computing Machinery, New York, NY, USA, 241–252. https://doi.org/10.1145/1831708.1831738

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. 712–721. https://doi.org/10.1109/ICSE.2013.6606617

[4] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*. 931–940. https://doi.org/10.1109/ICSE.2013.6606642

[5] Titus Barik, Brittany Johnson, and Emerson Murphy-Hill. 2015. I Heart Hacker News: Expanding Qualitative Research Findings by Analyzing Social News Websites. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 882–885. https://doi.org/10.1145/2786805.2803200

[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 (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 202–211. https://doi.org/10.1145/2597073.2597082

[7] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 188–197. https://doi.org/10.1109/ISSRE.2013.6698918

[8] Alan Blackwell, Alex McLean, James Noble, and Julian Rohrhuber. 2014. Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports* 3, 9 (2014), 130–168. https://doi.org/10.4230/DagRep.3.9.130

[9] Marion Blatt and Anthony Norman. 2013. *Email, communication and more: How software engineers use and reflect upon email at the workplace.* Master's thesis.

[10] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software* 146 (2018), 112–129. https://doi.org/10.1016/j.jss.2018.09.016

[11] David Boud and Heather Middleton. 2003. Learning from others at work: communities of practice and informal learning. *Journal of Workplace Learning* 15, 5 (2003), 194–202. https://doi.org/10.1108/13665620310483895

[12] Chris Brown and Chris Parnin. 2019. Sorry to Bother You: Designing Bots for Effective Recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 54–58. https://doi.org/10.1109/BotSE.2019.00021

[13] Chris Brown and Chris Parnin. 2020. Comparing Different Developer Behavior Recommendation Styles. In *Proceedings of the 13th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, ACM, 8.

[14] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. 2015. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 550–554. https://doi.org/10.1109/SANER.2015.7081875

[15] Antônio Carvalho, Welder Luz, Diego Marcílio, Rodrigo Bonifácio, Gustavo Pinto, and Edna Dias Canedo. 2020. C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 161–171. https://doi.org/10.1109/SANER48275.2020.9054842

[16] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2019. Building an Expert Recommender Chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 59–63. https://doi.org/10.1109/BotSE.2019.00022

[17] Alistair Cockburn and Laurie Williams. 2001. Extreme Programming Examined. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Chapter The Costs and Benefits of Pair Programming, 223–243. http://dl.acm.org/citation.cfm?id=377517.377531

[18] Brian de Alwis and Gail C. Murphy. 2006. Using Visual Momentum to Explain Disorientation in the Eclipse IDE. In *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE Press, Brighton, UK, 51–54. https://doi.org/10.1109/VLHCC.2006.49

[19] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (July 2019), 62–70. https://doi.org/10.1145/3338112

[20] David Evans and David Larochelle. 2002. Improving security using extensible lightweight static analysis. *IEEE software* 19, 1 (2002), 42–51. https://doi.org/10.1109/52.976940

[21] Jean-Marie Favre, Jacky Estublier, and Remy Sanlaville. 2003. Tool adoption issues in very large software company. In *3rd Workshop on Adoption Centric Software Engineering, ACSE*.

[22] Gerhard Fischer, Andreas Lemke, and Thomas Schwab. 1984. Active help systems. In *Readings on Cognitive Ergonomics — Mind and Computers*, Gerrit C. van der Veer, Michael J. Tauber, Thomas R. G. Green, and Peter Gorny (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 115–131. https://doi.org/10.1007/3-540-13394-1_10

[23] GitHub. 2019. The State of the Octoverse. https://octoverse.github.com/.

[24] 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 (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[25] Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-Based Development Research. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 368–371. https://doi.org/10.1145/2597073.2597122

[26] G. Gousios, A. Zaidman, M. Storey, and A. v. Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 358–368. https://doi.org/10.1109/ICSE.2015.55

[27] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment Analysis of Commit Comments in GitHub: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 352–355. https://doi.org/10.1145/2597073.2597118

[28] Sarah Heckman and Laurie Williams. 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology* 53, 4 (2011), 363 – 387. https://doi.org/10.1016/j.infsof.2010.12.007 Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[29] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2017), 547–578. https://doi.org/10.1007/s10664-016-9436-6

[30] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[31] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 92–101. https://doi.org/10.1145/2597073.2597074

[32] Lucas Layman, Laurie Williams, and Robert St. Amant. 2007. Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 176–185. https://doi.org/10.1109/ESEM.2007.11

[33] 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 Press, Nara, Japan, 386–395. https://doi.org/10.1109/APSEC.2018.00053

[34] Frank Linton, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. 2000. OWL: A recommender system for organization-wide learning. *Educational Technology & Society* 3, 1 (2000), 62–76.

[35] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (Sept. 2014), 37 pages. https://doi.org/10.1145/2622669

[36] Carlos Maltzahn and David Vollmar. 1994. *ToolBox: a living directory for Unix tools owned by the community.* Technical Report. Citeseer.

[37] Mika V. Mäntylä and Casper Lassenius. 2008. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering* 35, 3 (2008), 430–448. https://doi.org/10.1109/TSE.2008.71

[38] Nora McDonald and Sean Goggins. 2013. Performance and Participation in Open Source Software on GitHub. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*. Association for Computing Machinery, New York, NY, USA, 13–144. https://doi.org/10.1145/2468356.2468382

[39] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. In *Proceedings of the 11th*

Working Conference on Mining Software Repositories (MSR 2014). Association for Computing Machinery, New York, NY, USA, 192–201. https://doi.org/10.1145/2597073.2597076

[40] Justin Middleton, Emerson Murphy-Hill, Demetrius Green, Adam Meade, Roger Mayer, David White, and Steve McDonald. 2018. Which Contributions Predict Whether Developers Are Accepted into Github Teams. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. ACM, New York, NY, USA, 403–413. https://doi.org/10.1145/3196398.3196429

[41] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 84–94. https://doi.org/10.1109/ASE.2017.8115621

[42] Emerson Murphy-Hill. 2012. Continuous social screencasting to facilitate software tool discovery. In *2012 34th International Conference on Software Engineering (ICSE)*. 1317–1320. https://doi.org/10.1109/ICSE.2012.6227090

[43] Emerson Murphy-Hill, Da Young Lee, Gail C. Murphy, and Joanna McGrenere. 2015. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work (CSCW)* 24, 5 (2015), 389–422. https://doi.org/10.1007/s10606-015-9230-9

[44] Emerson Murphy-Hill and Gail C. Murphy. 2011. Peer Interaction Effectively, Yet Infrequently, Enables Programmers to Discover New Tools. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work (CSCW '11)*. ACM, New York, NY, USA, 405–414. https://doi.org/10.1145/1958824.1958888

[45] Gustavo H. Pinto and Fernando Kamei. 2013. What Programmers Say about Refactoring Tools? An Empirical Investigation of Stack Overflow. In *Proceedings of the 2013 ACM Workshop on Workshop on Refactoring Tools (WRT '13)*. Association for Computing Machinery, New York, NY, USA, 33–36. https://doi.org/10.1145/2541348.2541357

[46] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of GitHub README files. *Empirical Software Engineering* 24, 3 (2019), 1296–1327. https://doi.org/10.1007/s10664-018-9660-3

[47] Python. 2001. PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/#whitespace-in-expressions-and-statements.

[48] Martin P. Robillard and Christoph Treude. 2020. Understanding Wikipedia as a Resource for Opportunistic Learning of Computing Concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 72–78. https://doi.org/10.1145/3328778.3366832

[49] Martin P. Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation Systems for Software Engineering. *IEEE Software* 27, 4 (July 2010), 80–86. https://doi.org/10.1109/MS.2009.161

[50] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software Engineering at the Speed of Light: How Developers Stay Current Using Twitter. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 211–221. https://doi.org/10.1145/2568225.2568305

[51] Devarshi Singh, Varun Ramachandra Sekar, Kathryn T. Stolee, and Brittany Johnson. 2017. Evaluating how static analysis tools can reduce code review effort. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 101–105. https://doi.org/10.1109/VLHCC.2017.8103456

[52] Justin Smith, Chris Brown, and Emerson Murphy-Hill. 2017. Flower: Navigating program flow in the IDE. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Press, Raleigh, NC, 19–23. https://doi.org/10.1109/VLHCC.2017.8103445

[53] Will Snipes, Anil R. Nair, and Emerson Murphy-Hill. 2014. Experiences Gamifying Developer Adoption of Practices and Tools. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 105–114. https://doi.org/10.1145/2591062.2591171

[54] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (R) Evolution of Social Media in Software Engineering. In *Proceedings of the on Future of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 100–116. https://doi.org/10.1145/2593882.2593887

[55] Konstantinos Stroggylos and Diomidis Spinellis. 2007. Refactoring–Does It Improve Software Quality?. In *Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007)*. 10–10. https://doi.org/10.1109/WOSQ.2007.11

[56] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding Sparkle to Social Coding: An Empirical Study of Repository Badges in the Npm Ecosystem. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 511–522. https://doi.org/10.1145/3180155.3180209

[57] Rijnard van Tonder and Claire Le Goues. 2019. Towards s/engineer/bot: Principles for Program Repair Bots. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 43–47. https://doi.org/10.1109/BotSE.2019.00019

[58] Petcharat Viriyakattiyaporn and Gail C. Murphy. 2009. Challenges in the user interface design of an IDE tool recommender. In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. 104–107. https://doi.org/10.1109/CHASE.2009.5071421

[59] Petcharat Viriyakattiyaporn and Gail C. Murphy. 2010. Improving Program Navigation with an Active Help System. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '10)*. IBM Corp., USA, 27–41. https://doi.org/10.1145/1923947.1923951

[60] Anthony I. Wasserman. 1981. User Software Engineering and the Design of Interactive Systems. In *Proceedings of the 5th International Conference on Software Engineering (ICSE '81)*. IEEE Press, 387–393.

[61] 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. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 182. https://doi.org/10.1145/3274451

[62] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*. Association for Computing Machinery, New York, NY, USA, 1095–1106. https://doi.org/10.1145/2531602.2531722

[63] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan 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. https://doi.org/10.1109/MSR.2015.42