Comparing Different Developer Behavior Recommendation Styles

Chris Brown, Chris Parnin
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
{dcbrow10,cjparnin}@ncsu.edu

ABSTRACT

Research shows that one of the most effective ways software engineers discover useful *developer behaviors*, or tools and practices designed to help developers complete programming tasks, is through human-to-human recommendations from coworkers during work activities. However, due to the increasingly distributed nature of the software industry and development teams, opportunities for these peer interactions are in decline. To overcome the deprecation of peer interactions in software engineering, we explore the impact of several system-to-human recommendation systems, including the recently introduced *suggested changes* feature on GitHub which allows users to propose code changes to developers on contributions to repositories, to discover their impact on developer recommendations. In this work, we aim to study the effectiveness of suggested changes for recommending developer behaviors by performing a user study with professional software developers to compare static analysis tool recommendations from emails, pull requests, issues, and suggested changes. Our results provide insight into creating systems for recommendations between developers and design implications for improving automated recommendations to software engineers.

KEYWORDS

software engineering, developer behavior, developer recommendations, tool adoption

ACM Reference Format:
Chris Brown, Chris Parnin. 2020. Comparing Different Developer Behavior Recommendation Styles. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3391481

1 INTRODUCTION

Recommendations between peers is essential for workers to gain knowledge and improve the quality of their work. Boud and colleagues note that adults predominantly learn from others at work [7]. Software engineering is no exception, as developers frequently learn and share information with each other. For example, Cockburn and Williams found that learning is one of the benefits of *pair programming*, the practice of two developers writing code together at the same computer [11]. Furthermore, research shows that *peer interactions*, or the process of learning from colleagues in-person during normal work activities, are useful for software engineering tool recommendations [32, 33] and improving program comprehension [27]. Prior work suggests peer interactions are effective for developer recommendations because of their ability to incorporate receptiveness into suggestions for users [8].

While in-person recommendations are effective for increasing knowledge between software engineers, they are becoming less prevalent in the software industry. For example, even though Murphy-Hill and colleagues found peer interactions are an effective mode of tool discovery, he also discovered that these face-to-face recommendations occur infrequently in the workplace [33]. Furthermore, other work in software engineering points to a decrease in peer-to-peer interactions for programmers. Research shows open spaces designed to increase developer collaboration are unpopular among software engineers and decrease communication and productivity [39]. Additionally, the increase of distributed development teams and global software engineering has led to challenges in communication between developers [22].

To overcome the growing distribution of developers and decreasing opportunities for peer interactions, researchers have explored creating Recommender Systems for Software Engineering (RSSEs) to improve recommendations to developers [38]. However, studies such as [9] and [47] show that developers often find automated recommendations ineffective and usually do not adopt suggestions from these systems. Recently, GitHub introduced a new system for recommendations between developers called *suggested changes* [18]. This feature allows developers to collaborate by proposing code changes to contributors to improve code submitted to projects in pull requests. While suggested changes are increasing in popularity on GitHub, little is known about their impact on developer recommendations. To discover the impact of the suggested changes feature on knowledge sharing between GitHub users, we seek to answer the following research question:

RQ How well does the suggested changes feature generalize to different styles of recommendations?

To answer this research question, we conducted a user study evaluating static analysis tool recommendations with the suggested changes feature, a use case different from its intended purpose. We observed this to strictly evaluate the design of this feature for making recommendations to software engineers. Our results show that, compared to systems with different recommendation styles,

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391481

programmers prefer suggested changes for developer recommendations. The goal of this work is to discover the impact of suggested changes on knowledge sharing between developers and to provide implications for improving recommendations to software engineers. The main contribution of this work is the first study, to our knowledge, to examine the GitHub suggested changes feature.

2 BACKGROUND

2.1 Suggested Changes

Suggested changes allow users to recommend improvements to code on developers' pull requests. Figure 1 presents how the suggested changes feature works. After a user reviewing a pull request notices a line of code that can be improved, they can 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 a line on the pull request into a text box. Once the reviewer is finished with their suggestion, they can click on the "Start a review" button to submit the suggested change. Finally, the contributor can see the suggested change on their pull request, which is shown in Figure 1b. From here, the developer can commit the change, edit the suggestion, or ignore the proposed modification. If the developer accepts the change, the suggestion will automatically be integrated into their pull request as a new commit.

The GitHub blog reports users have been "quick to adopt suggested changes" into the code review process for their projects and over 100,000 uses within weeks of the initial public beta release, accounting for 4% of pull request comments and 10% of code reviewers during that time [19]. We examined the GitHub suggested changes feature because it allows developers to make recommendations to each other and have become very popular in the GitHub development community. The design of this feature allows code reviewers to make suggestions to programmers and allows developers to quickly and easily apply, reject, or edit the changes recommended by reviewers. The popularity of suggested changes show this feature is useful for recommendations between peers in the context of code reviews and proposing code improvements. We aim to evaluate the design of this novel feature for other types developer recommendations to provide implications for the design of future recommendation systems.

2.2 Developer Recommendations

Prior work has explored ways to make effective recommendations to improve *developer behaviors*, or useful tools and practices to help software engineers complete tasks more efficiently. Research shows that in-person interactions between programmers are effective for developer recommendations. For example, Cockburn and Williams suggest learning and sharing knowledge between developers is a primary benefit of pair programming [11]. Additionally, 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 normal work activities, were the most effective mode of tool discovery [32]. Likewise, Maaleej also shows peer interactions are effective for improving code comprehension [27].

Software engineering researchers have also created and evaluated many systems to make recommendations to assist programmers. Fischer and colleagues argue that *active help systems* are more effective for making recommendations to users completing tasks compared to *passive help systems*, which require users to explicitly seek help [14]. Robillard and colleagues discuss the importance of Recommender Systems for Software Engineering (RSSEs) for improving the decision-making of developers [38]. Prior work has introduced a variety of tools to recommend many tools and practices to developers to aid in the completion of programming tasks. For example, Spyglass is a recommender system that makes suggestions to help developers navigate code more efficiently in Eclipse [48].

Furthermore, research has also explored improving the human aspects of recommender systems. McNee and colleagues argue user-centric recommendations and experiences are more important for suggestions than accuracy [29]. Similarly, Konstan and colleagues suggest evaluating systems based on user experience metrics is more important than optimizing recommendation algorithms [25] while Murphy discovered software engineers find trust more important than precision in recommender systems [31]. In this work, we explore the suggested changes feature on GitHub to gain insight into designing effective systems for developer recommendations.

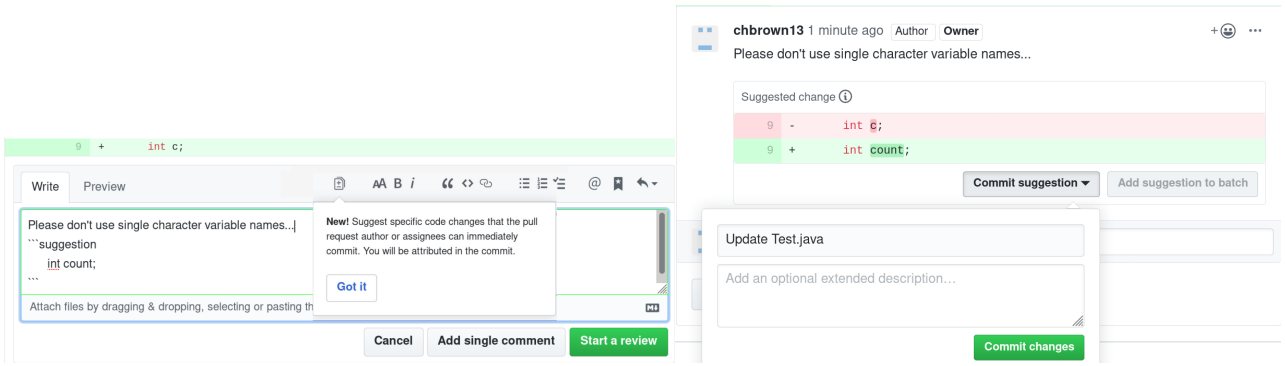
3 METHODOLOGY

3.1 Data Collection

To answer our research question, we conducted a user study to examine using the suggested changes feature for static analysis tool recommendations. We observed tool recommendations because, while studies shows static analysis tool usage is beneficial for software engineering teams, research also suggests developers rarely use them in practice [24]. 14 professional developers, presented in Table 1, participated in this study averaging 5 years of industry experience and working in various roles such as Software Engineer, Quality Engineer, Consultant, Data Migration Consultant, Support Specialist, User Researcher, and Technical Test Lead. We conducted an interactive think aloud study for participants to provide feedback on receiving recommendations with suggested changes.

3.2 Study Design

To determine the impact of suggested changes on developer recommendations, we asked participants to interact with static analysis tool recommendations from suggested changes, pull requests, issues, and emails. Each participant evaluated recommendations from all four systems simulated in an experimental repository they were not familiar with for our study. Tool adoption is a developer behavior beneficial for improving code quality and aiding programmers in their work [3]. Each recommendation style contained similar text recommending a static analysis tool to participants from each system and slightly differed in the presentation. For example, Figure 2 presents the suggested change recommendation which recommends a tool to find and prevent coding errors and suggests a fix for a reported bug. We selected these systems to compare with suggested changes based on prior work showing they are effective for recommendations:



(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

Email. Email is one of the most popular forms of communication today, with approximately 4 billion users who send and receive over 293 billion emails every day [45]. Sterne and colleagues suggest emails are the most powerful tool for reaching out and spreading information to target audiences for marketing [43]. In software engineering, Murphy-Hill and colleagues propose using email notifications to deliver tool recommendations to developers [34]. For example, the Coverity static analysis tool can alert developers of potential code defects via email [12].

Pull Requests. Pull requests allow developers to propose changes to projects [16]. We examined pull requests because they are the most popular method to recommend code changes to GitHub projects. In 2019, there were over 200 million pull requests submitted and 87 million pull requests were merged into repositories on the site [17]. Pull requests are also used to make suggestions to repositories on GitHub. Research shows that pull requests recommending enhancements to projects are the most common type of PR submitted by contributors and merged into repositories [36]. Additionally, prior work has explored generating automated pull requests to encourage developers to update package dependencies [30].

Issues. Issues are used track different types of information for GitHub repositories [15]. Over 20 million issues were closed by developers on GitHub in 2019 [17]. Additionally, we examined issues in this study because they are another method for users to make recommendations to peers on the site. Bissyandé and colleagues explored the GitHub issue tracker and found that, while the majority of issues with tags were labeled as "bugs", those labeled as "feature" or "enhancement" are "equally important for issue reporters" [6] while Krishna and colleagues also observed correlations between issues and enhancements added to projects and forecasting using predictive modeling [26]. Furthermore, Izquierdo and colleagues implemented GiLA to analyze OSS project issue labels [23].

3.3 Data Analysis

Participants interacted with each recommendation using a sample GitHub repository, and were asked to interact with each one as if they received it for their own project. Then, we asked them to provide a five point Likert-scale rating on how likely they would adopt the tool recommended from each system, to discuss what they like and dislike about each system, and provide general insight into what makes an effective recommendation during a semi-structured interview. User study participants are indicated with a P-prefix. Study sessions were audio and screen recorded. Two researchers performed an *open card sort* to develop categories from participant responses to gain insight for improving recommendations to software engineers [5]. We grouped statements into themes based on responses to what makes effective and ineffective recommendations then analyzed and discussed our themes to sort the data into five categories.

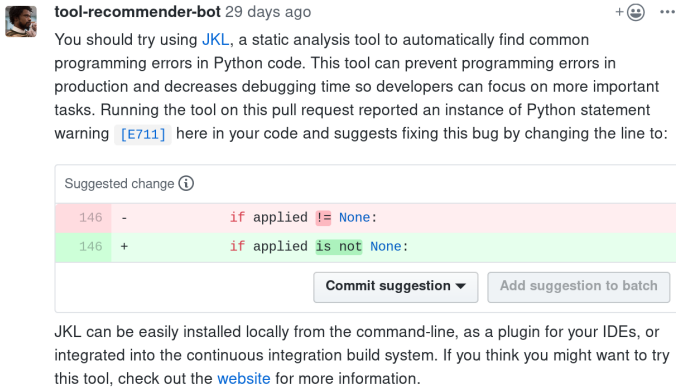


Figure 2: Mock Recommender system with suggested changes

4 RESULTS

4.1 Participant Scoring

Based on the results from our user study, we found that suggested changes were the preferred tool recommendation system by developers. Table 2 shows the average and median Likert scores representing the likelihood participants would adopt the tool from each method of recommendation. The Kruskal-Wallis test was used to

Participant	Experience (years)	GitHub Familiarity	OSS Contribution Frequency	Tool Usage Frequency
P1	30	Very Familiar	Occasionally	Very Frequently
P2	Less than 1	Moderately Familiar	Never	Never
P3	Less than 1	Very Familiar	Rarely	Moderately Frequent
P4	8	Very Familiar	Very Frequently	Very Frequently
P5	10	Familiar	Rarely	Moderately Frequent
P6	5	Moderately Familiar	Occasionally	Very Frequently
P7	6	Familiar	Frequently	Very Frequently
P8	6	Familiar	Very Frequently	Very Frequently
P9	Less than 1	Moderately Familiar	Occasionally	Very Frequently
P10	1	Moderately Familiar	Occasionally	Very Frequently
P11	3	Familiar	Very Frequently	Very Frequently
P12	3	Familiar	Rarely	Very Frequently
P13	1	Moderately Familiar	Never	Never
P14	1	Moderately Familiar	Never	Frequently

Table 1: User Study Participants

statistically measure differences in developer responses to tool recommendations from suggested changes, pull requests, issues, and email. We found a significant difference in likelihood of adoption provided by participants for the four systems for recommendations (Kruskal-Wallis, $H = 16.7527$, $p = .00079$, $\alpha = .05$). This shows that the suggested changes feature is not only effective for recommending code improvements on pull requests, but developers also find it effective for other types of recommendations, such as suggestions for static analysis tools.

	Average Score	*Median*
Suggestions	4	4
Pull Requests	3.71	4
Issues	2.86	3
Email	2.36	2

Table 2: Effectiveness for Developers

4.2 Qualitative Feedback

The user study was also a think aloud study and consisted of a semi-structured interview, where we were particularly interested in gaining insight into what developers believe makes an effective recommendation based on their interactions with the suggested changes feature compared to emails, pull requests, and issues, as well as general recommendation systems.

Emails. The majority of participants rated static analysis tool recommendations via email with a 1 or 2 ($n = 11$). Additionally, most developers provided hostile feedback on email recommendations such as "*I hate emails*" (P3), "*if this came across unsolicited I would feel sort of intruded upon*" (P4), "*would honestly be pretty suspicious when I get any email asking to install software on my computer*" (P12), "*if I see an email about something it actually gives me less of a view of it*" (P6), and "*I'd immediately delete it...I wouldn't even give it a look. I'd actually probably not like that tool even more just because their sending out spam emails*". However, two participants did rate email recommendations with a 5. In these cases, users preferred email because "*it feels personal*" (P2) and "*I like email more*" (P11).

Issues. Most participants were noncommittal on static analysis tool issue recommendations, with nine participants rating them with a 3. The primary disadvantage provided for issues was that a lack of examples and the amount of work it would take to learn more about the tool. For example, P1 noted "*I'd be much less likely to integrate it [the tool]...that's a lot of work*", P4 stated "*I see this as a big time sync to go through and evaluate how many of those actually are issues and how many are false positive things... just an example of a few of them would make this more appealing*", and P14 added "*it reads a little more spammy without a code example...It seems like you could just post this message on any project. Why is this useful for my project specifically? I have no idea*". Additionally, P13 complained "*this one is a ton of words*" about the issue recommendation.

Pull Requests. Developers were more interested in pull request recommendations, with ten participants rating them with a 3 or 4. Participants were primarily interested in the ability to easily integrate tools. For example, P4 noted with the pull request recommendations, developers are "*getting pretty quickly an explanation, the actual issue in the code, and you know a basically free way to incorporate that into the process as well as you know the tool itself*". Additionally, P7 said "*I'd be significantly more likely to try it if I already have a pull request that has all the changes I need to get the tool or something going in the project*". Detractors to pull requests mentioned the lack of information, such as P13 saying "*it doesn't really outline the steps*", and the method of using pull requests, such as P8 stating PRs should "*solve a problem rather than tell people they should try to use this thing*" and P14 adding "*it's not that they really want to do a pull request. It's not that their going to be adding to the project...I don't know why it's a pull request specifically*".

Suggested Changes. Participants significantly preferred static analysis tool recommendations with suggested changes during code reviews. 12 developers rated these suggestions with a 4 or 5. There were a variety of reasons participants favored suggested change recommendations. P8 praised the location of the recommendation saying, "*having it comment right here, 'This fixes your bug!' That's nice*". Participants liked the visual aspects of the feature, for example P13 noted that "*this one is definitely better because it's more visual*".

Developers also liked the details and information provided. For example, P9 noted "*it has a little more detail*" and P10 mentioned "*this has very good detail, very detailed change of the code and so it's clear*". One participant rated this feature with a 2 saying, "*putting a tool recommendation in a comment of a pull request, it's kind of to me out of place*".

4.3 General Feedback

In addition to collecting feedback from participants on each recommendation system, we also asked them to provide general insight into what they find makes recommendations effective. Below we present the five categories from our open card sort to describe how software engineers perceive recommendations and provide example quotes from participants:

Examples: Participants mentioned examples of how to use tools are important before making decisions on adoption. Developers desired to see examples of tool usage and output, view demos and test tools locally. Additionally, respondents desired the ability to easily access examples in the documentation and on the tool's website.

> *"In general I think showing an example of the type of error that it would find, cause that immediately shows some value, I think that helps a lot"* (P4)

Integration: This category refers to how well tools integrate into the workflow of developers. Participants noted how easy it is to install the tool, how well it works with other tools and build systems, the impact on resources (i.e. memory, GPU, lagging, etc.), how relevant it is to their project and needs, and if it adheres to company policies and mandates for tools are important considerations for adoption.

> *"I want something that I can install it and use it as quickly as possible with as minimal fussing with it and setup as possible."* (P5)

Marketing: For recommendations, developers noted that text sounding like advertising or marketing notifications would deter them from adopting suggestions. Responses were wary of emails, critical of unsolicited messages, and desired human-like communication.

> *"I'm not sure how the bot would generate the text to do the recommendation but try to make it seem a little more human? Rather than it was written by someone in an advertising agency or something."* (P7)

Popularity: Information about the popularity of tools was a very prominent factor for developers in making decisions on tool adoption. Participants noted seeking reviews, peers, social media and other resources before accepting recommendations.

> *"Try to highlight the popularity, popularity is so crucial...I care about the adoption. The current adoption, that's a testimony of the strength"* (P13)

Reliability: This refers to how trustworthy developers find tools and their sources. Participants stated that reliable tool output and usage as well as active development on the tool itself impact if recommendations are effective.

> *"[It] definitely needs to work reliably. Any time a tool starts doing things like it occasionally has problems that's something that makes me want to stop using it...I want the tool to be more reliable than that."* (P8)

5 DISCUSSION

Our results suggest that the GitHub suggested changes feature is an effective system for making recommendations to developers. Our results show developers preferred development tool recommendations with this feature over recommendations from other systems. Based on our evaluation, we use our results to provide implications for software engineers to make effective recommendations to colleagues and to software engineering researchers for designing systems for developer recommendations. To overcome the "alone together" recommendation barrier, we provide implications for improving developer recommendations based on feedback from developers on recommendation content and design.

5.1 Recommendation Content

Many participants noted that the content presented in recommendations impacts their decision on whether or not to adopt the target behavior. Here, we discuss the themes within recommendations themselves that encourage developers to adopt new behaviors. Based on feedback from participants, developers are more likely to adopt recommendations that easily integrate into their workflow and are popular among the software engineering community.

5.1.1 Workflow Details. Our results show that the impact of tools on workflow plays a major role in influencing the adoption. Other software engineering literature also shows that workflow integration is a key factor in developer adoption. For example, Brown and colleagues found that developers ignored automated tool recommendations because did not integrate well into their workflow, i.e. breaking CI builds [9]. Furthermore, Tonder and colleagues argue that successful bot integration with human workflows is important for improving their effectiveness [46]. We propose incorporating information about workflow impact into recommendations to improve recommendations and increase adoption of useful developer behaviors. To provide context for this concept, we discuss Relevance and Integration.

Integration. Our results show that the ability to integrate recommended tools into existing workflows and processes also impacts the effectiveness of recommendations. For example, P1 desired "*simple integration*". Furthermore, participants mentioned "*I want something that I can install it and use it as quickly as possible with as minimal fussing with it and setup as possible*" (P5) and "*ideally, I want it to be super easy to add to whatever CI or test runner system I've got*" (P8). Wasserman outlines tool integration techniques for software engineering environments [49]. Additionally, P6 also expressed a desire to demo tools before adoption, which could help improve integration. Furthermore, software engineering research shows the ability to integrate tools impacts usage, such as Favre and colleagues work outlining challenges for tool adoption in large software companies [13]. To incorporate this concept into developer recommendations, suggestions should provide information on integration such as how to install the tool, how to integrate it into CI build systems, whether it has a plugin for certain IDEs, etc.

Relevance. Furthermore, an important concept for integration is how relevant tools are to developers' work. Study participants provided statements such as "*it depends on if it's something I really need or want*" (P2), "*the recommendation itself matters less than how much I need the thing...If I don't need it then I'm not going to try it*" (P9), and "*I mainly do a lot of web development and data stuff. I don't really care about a new runtime for a new platform I'm never going to see I'm my work*" (P12). Developers also noted the relevance of information presented in our suggested changes recommendation, such as P8 stating "*Having it comment right here this fixes your bug? That's nice*". Prior work in recommender systems for software engineering also incorporates relevance into suggestions. For example, ToolBox is a community constructed recommender system that uses logged actions from users over a shared network to recommend relevant Unix commands [28]. To create effective recommendations, systems should note how products and behaviors are relevant to software engineers and improving their work.

5.1.2 Social Perspectives. Social perspectives in recommendations refers focusing on aspects of community in suggestions to potential users. In our results, we found most participants consider a variety of social aspects when deciding whether or not to adopt new tools. Ahmadi argues software engineering is a social activity [2], while Begel and colleagues explore the impact of social media on collaboration and knowledge sharing between software engineers [4]. Prior work suggests that community facets of software engineering can impact developer behavior and performance. For example, Steglich and colleagues show that social factors of software engineering impact mobile software ecosystem developers [42] and Brown and colleagues found that the inability to adhere to social aspects of software engineering prevented developers from adopting tools recommended by a bot [9]. Our results suggest researchers should integrate social influences into recommender systems. To increase the effectiveness of these systems, we propose implementing social recommendations by focusing on popularity and other sources.

Popularity. This concept refers to the reputation and adoption of tools and practices. In our study, we found that most developers desired information about tools from other users when making decisions. For example, participants said "*one of the things I want to see is what other people think about it*" (P3), "*how many people use it...the popularity of the tool being used would influence me to try that*" (P6), "*when there's a buzz around a tool, that's when you know it's good and you know it's worth checking out*" (P12), and "*try to highlight the popularity, popularity is so crucial*" (P13). Participants also mentioned learning about tools from peers. For instance, P6 noted an effective recommendation is "*word of mouth and people that I actually trust who use it*". Prior work also shows popularity and reputation impact adoption of developer behaviors. For example, Aggarwal and colleagues found that popularity metrics on GitHub impact updates to repositories and contributions from collaborators [1]. To increase effectiveness of recommendations, researchers should consider incorporating popularity statistics such as users, downloads, social media followers and likes, etc. to encourage adoption.

Other Sources. Several developers noted they preferred to see tools in multiple sources before adoption. For instance, participants mentioned desiring to see recommended tools from more familiar sources such as Google, Twitter, StackOverflow or StackExchange, tool websites, meetup groups, conference talks, and more. P4 also noted "*I think also in the recommendation having a lot of references is helpful*", P1 added "*I sort of want to hear about it where I hear about other programming tools*", and P5 wanted to be able to "*Google the tool's name...[and get] a link on the first page*". Prior work also shows that external sources play a role in developers' decisions to make decisions. For example, Xiao and colleagues found that software engineers trust security tool recommendations from credible sources on the Internet as much as in person recommendations from peers [50]. To improve developer recommendations, we propose integrating additional details about the desired tool or behavior from multiple sources to help programmers make informed decisions and increase adoption.

5.2 Recommendation Design

In addition to content, our results suggest the design of recommendations also impacts the decision outcome. Based on feedback from developers, we propose design implications for improving the effectiveness developer recommendations.

Examples. When commenting on what makes effective recommendations in our study, participants mentioned the presence of examples is important. For example, P5 desired "*a website that had examples with how to run it*", P10 stated "*It would be better if they can show me some examples with some very clear results like this is something you can get with our tool*", and P14 noted "*specific examples...would be a lot more compelling*". However, developers did not want to have to "*go to their website and have to click through a million different links to get an example. And sometimes they don't even have examples*" (P5). Additionally, developers also desired the ability to test tools themselves saying, "*I would like to try it by myself first*" (P11) and "*I would test it out locally*" (P8). Valaer and colleagues also show that examples are important in software engineering for helping developers choose user interface development tools [44]. To improve the design of recommendations, messages should include examples of how to use tools and sample output for users to observe.

Marketing. In our user study, participants also expressed disdain for recommendations that sound non-human or like advertisements. For example, P5 mentioned "*I'm not somebody who likes to get unsolicited marketing stuff*" while P1 and P7 added "*email...there's so much stuff that comes through email*" and "*email is definitely a no*". Another example of this is that developers found recommendations with "*a ton of words*" and that "*way too lengthy*" (P13) to be ineffective. Meanwhile, P10 prefers recommendations "*just getting to the point, don't show some other useless stuff which may confuse the potential user*". Meanwhile, developers praised the suggested changes recommendation because it provides a "*nice, concrete error*" (P8). In general, P13 did not like lengthy recommendations because "*it will take me a long time to digest it*". Similarly, Cerezo and colleagues suggest implementing *user-driven communication* to improve developer chatbots [10]. The negative perception of marketing among developers can also be seen from the backlash received by the maintainers of the JavaScript Standard Style guide, linter, and formatting tool [41], who attempted to raise funds for

development by incorporating advertisements within the terminal.[1] When creating recommender systems, designers should create short and concise recommendations to developers.

6 LIMITATIONS

An internal threat to the user study is response bias from participants. To avoid bias from participants' past experiences with existing tools or preferred programming languages in our evaluation, tools were recommended with made-up names (ABC, DEF, GHI, and JKL) and varying programming languages (JavaScript, Java, and Python). The order each participant interacted with the recommendation systems was randomized to avoid order bias in our results. We also allowed participants to revise scores for previous recommendation systems in the study as they interacted with the other systems.

An external threat to the validity of our results is that we generalize what makes an effective recommendations for developers when in reality each person has their own preferences for what makes good suggestions. Our sample size is also small, so our results may not generalize to all GitHub users and software developers. Even though we attempted to recruit a diverse sample of participants based on various characteristics such as gender, race, and age, the majority of participants in our user study were White males between the ages of 25-34. However, we found the population of our participation sample (16% female, 31% non-White) was consistent with data from recent studies on women [35] and minorities [37] in the software industry. Additionally, these results may not generalize to other code hosting platforms, such as GitLab [20], and other open or closed source software applications.

7 FUTURE WORK

Further research involves developing recommendation systems that incorporate workflow details and social perspectives into developer recommendations. We envision implementing the prototype recommendation system from our evaluation in Figure 2 to automatically suggest fixes to programming errors and recommend static analysis tools to developers. To improve this system, we plan to add workflow details, i.e. providing information to users about how to install and integrate tools into existing processes. We also aim to include social perspectives in recommendation context by incorporating user reviews and adoption rates. Additionally, we aim to design systems with specific examples as well as user-centric text that does not sound like marketing ads. We also hope to expand this work by creating recommendations for improving other developer behaviors such as code reviews and testing.

Research could also explore automated recommendations to developers on platforms and programming communities other than GitHub, such as StackOverflow [40], Hacker News [21], and GitLab [20]. To account for individual developer preferences for effective suggestions, future work could explore creating customized recommendations to programmers based on characteristics such as recent development activity and experience. Furthermore, AI and machine learning techniques, such as collaborative filtering, can be applied with our implications into recommender systems to predict poor developer behaviors and proactively send recommendations.

[1] https://github.com/standard/standard/issues/1381

8 CONCLUSION

We analyzed the GitHub suggested changes feature, a system that allows users to recommend code changes to developers on pull requests, in the context of development tool recommendations. We evaluated recommendations with this feature by performing a user study with 14 professional software developers. Our results suggest software engineers find this system effective for development tool recommendations and significantly prefer it to other recommendation styles. To improve the effectiveness of future recommender systems, we encourage researchers to improve the content of suggestions by incorporating workflow details and social perspectives and improve the design of recommendations by including examples and using user-centric language.

9 ACKNOWLEDGEMENTS

We would like to thank all of the developers who took the time to participate in this study for their contributions. Additionally, we would also like to thank Samim Mirhosseini for his assistance with data analysis during the open card sorting process.

REFERENCES

[1] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. 2014. Co-Evolution of Project Documentation and Popularity within Github. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 360–363. https://doi.org/10.1145/2597073.2597120

[2] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic. 2008. A survey of social software engineering. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*. IEEE, 1–12. https://doi.org/10.1109/ASEW.2008.4686304

[3] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (Sep. 2008), 22–29. https://doi.org/10.1109/MS.2008.130

[4] Andrew Begel, Robert DeLine, and Thomas Zimmermann. 2010. Social Media for Software Engineering. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER '10)*. Association for Computing Machinery, New York, NY, USA, 33–38. https://doi.org/10.1145/1882362.1882370

[5] Andrew Begel and Thomas Zimmermann. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 12–23. https://doi.org/10.1145/2568225.2568233

[6] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. 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)*. IEEE, 188–197. https://doi.org/10.1109/ISSRE.2013.6698918

[7] 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.

[8] C. Brown, J. Middleton, E. Sharma, and E. Murphy-Hill. 2017. How software users recommend tools to each other. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 129–137. https://doi.org/10.1109/VLHCC.2017.8103460

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

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

[11] Alistair Cockburn and Laurie Williams. 2001. The Costs and Benefits of Pair Programming. In *Extreme Programming Examined*, Giancarlo Succi and Michele Marchesi (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 223–243. http://dl.acm.org/citation.cfm?id=377517.377531

[12] Coverity Static Application Security Testing [n. d.]. Synopsis. https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html.

[13] 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*. ACM, 81–89.

[14] Gerhard Fischer, Andreas Lemke, and Thomas Schwab. 1984. Active help systems. In *Readings on Cognitive Ergonomics—Mind and Computers*. Springer, 115–131.

[15] GitHub. [n. d.]. About issues. *GitHub Help* ([n. d.]). https://help.github.com/en/articles/about-issues

[16] GitHub. [n. d.]. Creating a pull request. *GitHub Help* ([n. d.]). https://help.github.com/en/articles/creating-a-pull-request

[17] GitHub 2018. The State of the Octoverse. https://octoverse.github.com/.

[18] GitHub. 2018. Suggested Changes (public beta). *The GitHub Blog* (2018). https://github.blog/changelog/2018-10-16-suggested-changes/

[19] GitHub. 2018. Suggested changes: what we've learned so far. *The GitHub Blog* (2018). https://github.blog/2018-11-01-suggested-changes-update/

[20] GitLab [n. d.]. GitLab. https://about.gitlab.com/.

[21] Hacker News [n. d.]. Y Combinator. https://news.ycombinator.com/.

[22] J. D. Herbsleb. 2007. Global Software Engineering: The Future of Socio-technical Coordination. In *Future of Software Engineering (FOSE '07)*. IEEE, 188–198. https://doi.org/10.1109/FOSE.2007.11

[23] Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi, Alexandre Bergel, and Jordi Cabot. 2015. Gila: Github label analyzer. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 479–483.

[24] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. 2. Conference. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE) (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[25] Joseph A Konstan and John Riedl. 2012. Recommender systems: from algorithms to user experience. *User modeling and user-adapted interaction* 22, 1-2 (2012), 101–123.

[26] Rahul Krishna, Amritanshu Agrawal, Akond Rahman, Alexander Sobran, and Tim Menzies. 2018. What is the connection between issues, bugs, and enhancements?: Lessons learned from 800+ software projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 306–315.

[27] 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

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

[29] Sean M. McNee, John Riedl, and Joseph A. Konstan. 2006. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06)*. ACM, New York, NY, USA, 1097–1101. https://doi.org/10.1145/1125451.1125659

[30] Samim Mirhosseini and Chris Parnin. 2017. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 84–94.

[31] Gail C. Murphy and Emerson Murphy-Hill. 2010. What is Trust in a Recommender for Software Development?. 4. Workshop. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE) (RSSE '10)*. ACM, New York, NY, USA, 57–58. https://doi.org/10.1145/1808920.1808934

[32] 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

[33] 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

[34] Emerson Murphy-Hill and Gail C Murphy. 2014. Recommendation delivery. In *Recommendation Systems in Software Engineering*. Springer, 223–242.

[35] Sohan Murthy. 2014. Women in Software Engineering: The Sobering Stats. *LinkedIn Talent Blog* (2014). https://business.linkedin.com/talent-solutions/blog/2014/03/women-in-engineering-the-sobering-stats

[36] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. A Study of External Community Contribution to Open-Source Projects on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 332–335. https://doi.org/10.1145/2597073.2597113

[37] Sinduja Rangarajan. 2018. Here's the clearest picture of Silicon Valley's diversity yet: It's bad. But some companies are doing less bad. *The Center for Investigative Reporting* (2018). https://www.revealnews.org/article/heres-the-clearest-picture-of-silicon-valleys-diversity-yet/

[38] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation systems for software engineering. *IEEE software* 27, 4 (2010), 80–86.

[39] Edward K. Smith, Christian Bird, and Thomas Zimmermann. 2016. Beliefs, Practices, and Personalities of Software Engineers: A Survey in a Large Software Company. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '16)*. Association for Computing Machinery, New York, NY, USA, 15–18. https://doi.org/10.1145/2897586.2897596

[40] Stack Overflow [n. d.]. Stack Exchange Network. https://stackoverflow.com/.

[41] StandardJS [n. d.]. JavaScript Standard Style. https://standardjs.com/.

[42] C. Steglich, S. Marczak, C. R. B. De Souza, L. P. Guerra, L. H. Mosmann, F. Figueira Filho, and M. Perin. 2019. Social Aspects and How They Influence MSECO Developers. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 99–106. https://doi.org/10.1109/CHASE.2019.00032

[43] Jim Sterne and Anthony Priore. 2000. *Email marketing: using email to reach your target audience and build customer relationships*. John Wiley & Sons, Inc.

[44] Laura A Valaer and Robert G Babb. 1997. Choosing a user interface development tool. *IEEE Software* 14, 4 (1997), 29–39.

[45] Jordie van Rijn. [n. d.]. ([n. d.]).

[46] Rijnard van Tonder and Claire Le Goues. 2019. Towards s/Engineer/Bot: Principles for Program Repair Bots. In *Proceedings of the 1st International Workshop on Bots in Software Engineering (BotSE '19)*. IEEE Press, 43–47. https://doi.org/10.1109/BotSE.2019.00019

[47] P. Viriyakattiyaporn and G. 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*. IEEE, 104–107. https://doi.org/10.1109/CHASE.2009.5071421

[48] 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

[49] 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.

[50] 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