Research Statement

Dwayne "Chris" Brown, Jr.
North Carolina State University

Research Overview

Decision-making plays an important role in software engineering, however developers often require help when making decisions. For example, software engineering researchers and toolsmiths create and evaluate a wide variety of tools and practices to help accomplish programming tasks, but these useful *developer behaviors* often go ignored by software engineers in industry. This problem can be summed up in the following question posed by Greg Wilson: ***"How do we get working programmers to actually adopt better practices?"***[1]

My research goal is, given a developer who is unaware of a useful developer behavior in a development situation, identify the most effective strategy to convince them to adopt the behavior. To encourage developers to adopt better practices, my dissertation research involves interdisciplinary work focused on analyzing how developers adopt behaviors through the lens of *nudge theory*, a behavioral science framework to improve human decision-making without providing incentives or banning selections [1].

My research experiences involve numerous undergraduate and graduate opportunities studying a variety of computer science topics. Overall, my research philosophy is two-fold: 1) performing empirical studies to collect and analyze data characterizing software engineering problems, and 2) to develop tools and techniques to help solve these problems. This is shown in my doctoral research contributions, which incorporates include several multimethodology studies analyzing qualitative and quantitative data to explore effective automated recommendations for improving developer behavior.

Research Contributions

My research contributions include a *conceptual framework* characterizing failures of developer recommendations, evaluating *existing tools*, and developing an *automated recommender system* that uses nudges to make effective recommendations for developer behaviors to software engineers.

[1] https://twitter.com/gvwilson/status/1142245508464795649?s=20

Characterizing failures of developer recommendations.

To characterize failures of developer recommendations, we analyzed *peer interactions* and developed a baseline naive *telemarketer design* approach.

Research suggests learning from colleagues during work activities, or *peer interactions*, is the most effective mode of tool discovery for developers. To discover what makes peer interactions effective, we conducted a user study observing tool recommendations between pairs of participants working together to complete data analysis tasks and conducted semi-structured interviews to explore recommendations between peers. For each study, we analyzed sessions to determine 1) when peer interactions occurred, 2) if interactions contained characteristics such as politeness, persuasiveness, receptiveness, and time pressure based on a set of criteria we developed for each, and 3) if recommended tools were used or ignored by users who received suggestions. Overall, we analyzed 142 peer interactions and found that *receptiveness* was the only significant characteristic that impacted peer interactions (Wilcoxon, $p = 0.0002$, $\alpha = .05$). To define receptiveness, we used the two criteria: ***Demonstrate Desire*** and ***Familiarity*** [2].

While peer interactions are effective for recommending tools, they do not scale with larger and distributed development teams. To create a baseline approach for automated recommendations, we developed the naive *telemarketer design* which "calls" users to deliver generic messages without the ability to customize recommendations or respond to users. We implemented this design in *tool-recommender-bot*, a system that generates automated pull requests recommending tools by using simple examples of common programming errors and changing build configuration files to add tools to projects. To evaluate our approach, we recommended the Error Prone Java static analysis tool[2] to 52 GitHub repositories. Ultimately, the naive *telemarketer design* in *tool-recommender-bot* was only able to make two effective recommendations (4%) while most PRs were closed or ignored. Based on comments made by developers, we found that our baseline approach failed because it lacked ***social context*** and poorly integrated into ***developer workflows*** [3].

In summary, we uncovered four concepts to consider for improving the effectiveness of recommendations to developers: **desire**, **familiarity**, **social context**, and **developer workflow**. Integrating these into automated recommendations can improve developer behavior and increase adoption of useful programming tools and practices.

[2]http://errorprone.info

Evaluating existing tools.

To analyze existing systems for making recommendations to developers, we evaluated the GitHub *suggested changes* feature. This recently introduced feature fits into the nudge theory framework and allows users to recommend code changes to developers on pull requests. To study GitHub suggestions, we conducted a multimethodology study divided into two phases. Phase 1 is an empirical study where we mined GitHub for instances of suggested changes to categorize types of changes developers propose and determine how often suggestions are accepted. Our results show that developers suggest *corrective*, *improvement*, *formatting*, and *non-functional* code changes. However, PRs are still the most effective recommendation system on GitHub (78%) compared to suggested changes (69%) and issues (17%). For Phase 2, we collected developer feedback by surveying users who received or made suggestions and conducting a user study with 14 professional developers exploring recommendations with this feature. We show that users found suggested changes useful primarily because they improve communication, provide concise changes, and save time. Additionally, we discovered developers significantly preferred tool recommendations from suggested changes over emails, PRs, and issues (Kruskal-Wallis, $p = .00079$, $\alpha = .05$). Based on our results, we found four design implications for developer recommender systems: *social recommendations*, *user-driven communication*, *actionable recommendations*, and *spatial* and *temporal locality* [4].

Developing an automated recommender system.

To incorporate the findings from my preliminary work into an automated recommender system, we developed *nudge-bot*, a bot designed to generate *active* and *passive* nudges encouraging developers to adopt better behaviors. To discover the impact of actionable recommendations on developer behavior, we will incorporate our system into an upper-level undergraduate Software Engineering course for their final project in Spring 2020. Data collection for this evaluation will consist of mining GitHub repositories from previous semesters to compare how these interventions impact student productivity, observing student behaviors for project management and collaboration, and collecting feedback from students on their perception of the automated recommendations. We hypothesize our system will improve developer behavior by: 1) increasing the number functional and process requirements completed for projects, 2) increasing collaboration for student contributions, 3) reducing procrastination, and 4) improving software quality and project grades.

Future Work

My career research goal is to continue exploring ways to improve the behavior and productivity of software engineers. The following research areas highlight future directions based on my research interests and prior work:

- **Developer Nudges:** One limitation of this work is that we generalize "developers" when exploring what makes effective recommendations. In reality, developers are extremely diverse each developer has individual preferences for what makes suggestions effective and what types of interventions are useful in certain situations. To create better strategies for encouraging developers to adopt useful practices, I aim to study customized *developer nudges*. To start this work, I plan to evaluate the effectiveness of different types of interventions to GitHub users based on characteristics such as recent development activity and experience.

- **Proactive Recommendations:** The recommendations implemented for my work so far have been *reactive*, in that they suggest behaviors after developers have started or completed programming tasks in a manner that could be improved. To improve the effectiveness of developer nudges, I plan to develop *proactive* recommendations to predict developer actions while completing coding tasks and suggest useful behaviors before developers start. To begin this work, I plan to apply machine learning techniques such as collaborative filtering into recommender systems to anticipate poor developer behaviors and send proactive nudges to recommend better practices.

- **Integrating Research in Industry:** Additionally, I am passionate about finding ways to bridge the gap between software engineering research and industry. To increase awareness of research tools and techniques for software engineers, I plan to continue to studying effective recommendation approaches for developer behaviors by expanding nudges to include additional practices and tools created and evaluated by the research community as well as integrating these behaviors into CS classes to instill in future software engineers. To increase the relevance of industry problems for research, I aim to pursue collaborations with industry developers, design empirical studies and tools that impact real-world practices, gather feedback from professional developers, and present results to industry developers in a way that increases awareness and motivates adoption of useful developer behaviors.

Broader Impact

My research has been published at peer-reviewed conferences and workshops including Visual Languages and Human-Centric Computing (VL/HCC), International Workshop on Bots in Software Engineering (BotSE), and International Conference on Software Engineering (ICSE). My work has also been presented at industry conferences such as Red Hat QECampX and DevConf. Finally, the tools and artifacts created for this research are open source and publicly available for developers and researchers to use, evaluate, and extend. This includes *tool-recommender-bot*,[3] *nudge-bot*,[4] and more.

References

[1] Cass Sunstein, Richard Thaler, et al. Nudge. *The politics of libertarian paternalism. New Haven*, 2008.

[2] ***Chris Brown***, Justin Middleton, Esha Sharma, and Emerson Murphy-Hill. "How software users recommend tools to each other". In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC 2019, pages 129–137, Raleigh, NC, USA, Oct 2017. IEEE Press.

[3] ***Chris Brown*** and Chris Parnin. "Sorry to bother you: Designing bots for effective recommendations". In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE 2019, pages 54–58, Montreal, QC, Canada, May 2019. IEEE Press.

[4] ***Chris Brown***, Zhe Yu, and Chris Parnin. "Understanding the impact of github suggested changes on recommendations between developers". Submitted to *Proceedings of the 42nd International Conference on Software Engineering*, ICSE 2020, Seoul, South Korea, May 2020. IEEE Press.

[3] https://github.com/chbrown13/tool-recommender-bot
[4] https://github.com/chbrown13/nudge-bot