Digital Nudges for Encouraging Developer Actions

Chris Brown
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
dcbrow10@ncsu.edu

***Abstract*—Researchers have examined a wide variety of practices to help software engineers complete different programming tasks. Despite the fact that studies show software engineering practices and tools created to improve the software development process are useful for preventing bugs, decreasing debugging costs, reducing debugging time, and providing additional benefits, software engineers rarely use them in practice. To persuade humans to alter and adopt new behaviors, psychologists have studied the concept of nudges. My research aims to investigate how digital nudges, or the process of using technology to automatically create nudges, can be beneficial in helping software developers and teams adopt software engineering activities and integrate them into their normal workflow.**

***Index Terms*—software engineering, developer actions, tool adoption, nudge theory**

I. Introduction

To help developers write higher quality code, researchers have created and evaluated a wide variety of useful practices to help in completing programming tasks. For example, toolsmiths have developed tools to aid developers in accomplishing important aspects of the software development process including code analysis (static and dynamic), debugging, refactoring, documentation, version control, continuous integration, communication, and security. Studies show development tools are useful for improving code [1], preventing errors [3], saving money by lowering debugging costs [15], reducing fault fix latency [14], or the time between a defect injection and it's removal, and making developers more efficient [11]. Other examples of software engineering activities for improving software quality are code reviews, testing, pair programming, and continuous integration.

However, despite the fact that research provides evidence these practices are beneficial for completing programming tasks during software development, developers rarely use them. For instance, continuing with the useful software engineering practice of integrating tools into the development process, prior work has examined barriers preventing software engineers from adopting tools for debugging [6], static analysis [12], security [34], refactoring [21], documentation [9], open source contribution [18], and build automation [25]. Additionally, Tilley and colleagues argue adoption is one of the most important goals of software engineering research, and present challenges for professional developers face with programming tools created by researchers, or research-off-the-shelf (ROTS) software [31]. There are many reasons developers ignore tools, yet this work focuses on the discoverability barrier where users are unaware that a useful tool exists [20]. Increasing awareness of software engineering practices, such tool adoption, can improve code quality and developer productivity in industry.

Psychology researchers have examined how to persuade humans to adopt new behaviors or ideas by studying choice architecture, or the context of how people make decisions [30]. There are many aspects of choice architecture that can change human behavior, and the one we chose to focus on is *nudge theory*. A *nudge* is defined as any factor in decision-making that can alter behavior without incentives or banning alternatives, i.e. the arrangement of possible options [29]. Furthermore, using technology to create nudges that modifies human behavior in digital environments is called to *digital nudging* [33]. We hypothesize utilizing digital nudges to send recommendations for development tools to programmers can increase awareness and adoption of these systems.

Nudging developers to adopt *software engineering actions*, or activities and processes designed to improve programming tasks, can help avoid poor-quality applications, inconvenienced users, increased debugging and maintenance costs, and wasted time and effort for developers. This research will solve the following problem: Given a developer who is unaware of an applicable software engineering tool, identify the most effective strategy for convincing them to adopt the tool. We created digital nudges aimed towards software engineers, and will evaluate their effectiveness by building and analyzing systems to evaluate their performance. The contributions of my research include:

- a *conceptual framework* building on research in nudge theory to improve software engineer behavior
- a set of *experiments* to evaluate and provide evidence to support our framework
- an *automated recommender system* to generate digital nudges recommending software engineering actions to developers

II. Related Work

My research is based on prior work that explores making recommendations to developers and technical approaches to increasing software engineering action adoption.

Recommendation: Research has investigated how to make effective recommendations to users in many different domains. For software engineers, Murphy-Hill examined different ways developers learn about tools and found that *peer interactions*, or learning about tools from peers during work activities, was the most effective [23]. Prior work has also proposed using concepts such as *diffusion of innovations* [26], *idea gardening* [5], and *information foraging theory* [24] to spread ideas and make effective recommendations to software developers. To our knowledge, our work is the first research to integrate nudge theory into recommendations for software engineers.

Adoption: Furthermore, prior work has proposed and evaluated methods to solve the software engineering adoption problem. Examples include using automated pull requests to encourage upgrading dependencies [19], screencasting to improve tool discovery [20], crowdsourcing to enhance coding tutorials [10], explorative and exploitative searching to recommend tasks to developers [13], logging to recommend Unix commands to users over a shared network [17], creating a knowledge base to monitor developer action history to improve code navigation [32], and gamification to encourage tool usage by rewarding developers with points towards a leaderboard for utilizing code navigation tools during development [2].

III. Research Plan

This section outlines our conceptual framework (Section III.A), completed study results (Section III.B), research in progress at the time of this writing (Section III.C), future projects (Section III.D), and research timeline (Section III.E).

A. Conceptual Framework

Nudge theory is a concept that has shown to modify human behavior in many different domains. Digital nudges are effective in altering behavior through digital choice environments, for example FitBit[1] activity trackers prodding users to increase physical activity and make healthier lifestyle decisions [33]. The creators of the term nudge provide two advantages to this type of recommendation, it's "easy and cheap" [29, p. 71]. Nudges are a viable option for making recommendations because they do not incentivize users and do not prohibit alternative options. This research aims to discover when developers are more likely to accept a recommendation based on different factors such as visibility in the context of their work, timeliness, trust, etc.

The conceptual framework will introduce and evaluate new types of digital nudges for software engineers. Example nudge types include: *social nudges* that suggest actions adopted by friends or colleagues; *public nudges* which are visible and may involve social pressure if behaviors aren't adopted; *just-in-time nudges* to make recommendations when a software engineering action is appropriate and *situated nudges* to make suggestions where the practice is applicable; *apprehensive nudges* that provide multiple locations where software engineering activities can be applied; *automated nudges* that perform actions for developers and present output; *tutorial nudges* which show developers how to complete a practice; *reminder nudges* to periodically recommend actions to programmers; and *positive nudges* that commend developers to encourage action adoption. This research will build new tools and examine existing systems that implement these digital nudge types, in the context of tool adoption, to evaluate their effectiveness in impacting developer behavior and actions.

B. Preliminary Findings

How Software Users Recommend Tools to Each Other

Motivation: Using development tools is a useful software engineering practice that the Software Engineering Body of Knowledge says can achieve "desirable characteristics of software products [28]. Previous software engineering research shows peer interactions are the most effective way to learn about tools [22], [23]. The goal of this study was to examine peer interactions in an experimental setting to better understand what makes user-to-user recommendations, a form of social nudge, effective for recommending tools. Our results provide implications for improving future automated recommendation approaches to recommend software engineering practices to developers.

Methodology: In this study, we designed an experiment to observe peer interactions and determine what makes them an effective method for learning about new tools in software. To observe peer interactions, we recruited students from North Carolina State University and professional workers from the Laboratory for Analytic Sciences[2] to work with a partner to complete data analysis tasks. This project sought to investigate the following research question:

> **RQ:** What characteristics of peer interactions make recommendations effective?

The characteristics we analyzed are politeness, persuasiveness, receptiveness, time pressure, and tool observability. These characteristics were motivated by Murphy-Hill's previous work on peer interactions and psychology research. Effectiveness was measured by noting instances where participants used or ignored suggestions when faced with opportunities to use a tool recommended by their partner during the study.

Results: We analyzed 142 total recommendations between participants and found that *receptiveness* was the only characteristic that significantly impacts the outcome of a tool recommendation between peers (Wilcoxon, $p = 0.0002$). Our results suggest recommender systems should target users' receptivity when making tool suggestions. To define receptiveness, we created criteria using Fogg's work on designing persuasive technology [8]. The criteria for receptiveness were: *Demonstrate Desire* and *Familiarity*. The implications of this work indicate that future automated recommendation approaches can make more effective recommendations by prioritizing user receptiveness and integrating these criteria. These results were published in the 2017 Visual Languages and Human-Centric Computing (VL/HCC) conference [4].

C. In Progress

tool-recommender-bot

Motivation: Research shows *active help systems* are more effective than passive ones requiring users to explicitly seek help [7]. To examine how developers respond to digital nudges,

[1]https://www.fitbit.com/home

[2]https://ncsu-las.org/

we developed *tool-recommender-bot*. *tool-recommender-bot* is an automated recommender system that makes suggestions to software engineers on GitHub, a popular code-hosting website. *tool-recommender-bot* differs from existing tool recommender systems, like [16], [17], [32], because it integrates a combination of digital nudge types and incorporates receptiveness, targeting the *desire* of programmers to write high-quality code and their *familiarity* with the code base, to suggest software engineering actions to developers.

To identify a baseline for effective automated recommendations, we conducted a preliminary evaluation using *tool-recommender-bot* to make simple recommendations to developers on 52 GitHub repositories. We found that only 4% of these suggestions were successful, and the majority of feedback and responses from developers were negative. Our results suggest that our naive recommendations were ineffective in influencing developer behavior because of their lack of social context and difficulty integrating into developers' workflow. These results are under review for submission to the International Workshop on Bots in Software Engineering[3] (BotSE) at ICSE 2019 at the time of this writing.

Methodology: We designed *tool-recommender-bot* to integrate all our digital nudge types and to be extendable for recommending a variety of software engineering actions to developers. Our initial implementation recommends ERROR PRONE, a static analysis tool for Java,[4] to increase static analysis tool adoption. In this study, we aim to improve on our naive recommendation design by implementing concepts from theory into recommendations sent to programmers from our system. To evaluate our system, our study will observe GitHub repositories, analyze changes made by developers, recommend ERROR PRONE as a comment on pull requests, and observe how developers respond to nudge types by placing a survey link at the end of the recommendation comment. The survey will consist of 5-point Likert scale and free response questions to gather qualitative and quantitative data on the effectiveness recommendations from *tool-recommender-bot*.

Hypothesis: We hypothesize that *tool-recommender-bot* recommendations incorporating our new digital nudge types will be more effective and useful for developers. For instance, one digital nudge type we plan to evaluate in this study are apprehensive nudges. We will study this by varying whether or not developers receive recommendations with other lines of code where ERROR PRONE reports a bug. Nudge theory shows people make better choices when they have good information in choice environments [29, p. 73].

GitHub suggestions

Motivation: GitHub recently introduced a new feature for users to suggest code changes to developers in lines changed during a pull request reviews.[5] Early analysis shows these situated nudges are very popular and effective among software engineers, already totaling over 100,000 uses with developers "quick to adopt suggested changes" and integrate them into the code review process.[6] This research aims to discover why GitHub suggestions, or situated nudges, are effective for influencing developer actions and behavior.

Methodology: To analyze suggestions, we developed a script to automatically analyze comments and search for suggested lines of code on pull requests for the most popular GitHub repositories. After gathering a corpus of code suggestions, we will send a survey to developers who received a code suggestion. The survey will be distributed via email to GitHub users who receive a suggestion with an email address publicly available on their profile. We will ask 5-point Likert scale questions on the usefulness of the GitHub suggestions feature and include open-ended free response questions for developers to add more information about why they find this system effective. Our evaluation will provide quantitative and qualitative data based on the responses from developers to characterize how developers respond to situated nudges.

Hypothesis: We hypothesize developers will find situated nudges effective because of their location within the code. Nudge theory research suggests location is important, for example the arrangement of food in a cafeteria or grocery store can have a major impact on the type of food people purchase and consume [29, p. 68]. Prior research in software engineering also shows in situ design is important for programming tasks, such as code navigation [27].

D. Future Work

tool-recommender-bot 2.0

Motivation: We designed *tool-recommender-bot* to make recommendations to developers using our new digital nudge types. With the results from our research studies on peer interactions, *tool-recommender-bot*, and GitHub suggestions, we plan to improve our system to make recommendations to software engineers. Future iterations of *tool-recommender-bot* will further our work by recommending additional software engineering tools, integrating different digital nudge types, and examining when each nudge type is most effective for making better suggestions to developers for specific situations and programming tasks.

Methodology: The first phase of this evaluation will conduct a similar experiment to the first *tool-recommender-bot* study with several changes. First, we plan to recommend a different software engineering tool other than ERROR PRONE to GitHub developers. Next, we will select additional digital nudge types to vary in experimental recommendations to developers, measuring effectiveness based on data gathered from surveys to developers. The second phase will analyze results from our research on all of the digital nudge types to create a framework describing when each nudge type is useful to developers. For example, the peer interactions study found that social nudges are most effective when users are receptive to a receiving a software engineering action recommendation.

[3]https://botse.github.io/

[4]http://errorprone.info/

[5]https://help.github.com/articles/incorporating-feedback-in-your-pull-request/#applying-a-suggested-change

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

E. Timeline

The author is a fourth year computer science PhD student at NC State University working with Dr. Chris Parnin. Table I presents the proposed research plan timeline:

Milestone	**Target**
Peer Interaction	VL/HCC 2017
Preliminary Results	BotSE* at ICSE 2019
tool-recommender-bot	FSE 2019*
Suggestions	VL/HCC 2019*
tool-recommender-bot2.0	ICSE 2020*
Defense	Spring 2020

TABLE I: Research Plan Timeline

* These milestones depend on paper acceptances into conferences. In case of rejection or delay, other potential venues for submission include VL/HCC, RecSys, CSCW, ASE, TSE, and other peer-reviewed conferences and journals.

REFERENCES

[1] N. Ayewah and W. Pugh. The google findbugs fixit. In *Proceedings of the 19th international symposium on Software testing and analysis*, pages 241–252. ACM, 2010.

[2] T. Barik, E. Murphy-Hill, and T. Zimmermann. A perspective on blending programming environments and games: Beyond points, badges, and leaderboards. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*, pages 134–142. IEEE, 2016.

[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, 53(2):66–75, 2010.

[4] C. Brown, J. Middleton, E. Sharma, and E. Murphy-Hill. How software users recommend tools to each other. In *Visual Languages and Human-Centric Computing*, 2017.

[5] J. Cao, I. Kwan, F. Bahmani, M. Burnett, S. D. Fleming, J. Jordahl, A. Horvath, and S. Yang. End-user programmers in trouble: Can the idea garden help them to help themselves? In *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 151–158, Sept 2013.

[6] J. Cao, K. Rector, T. H. Park, S. D. Fleming, M. Burnett, and S. Wiedenbeck. A debugging perspective on end-user mashup programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 149–156. IEEE, 2010.

[7] G. Fischer, A. Lemke, and T. Schwab. *Active help systems*, pages 115–131. Springer Berlin Heidelberg, Berlin, Heidelberg, 1984.

[8] B. Fogg. Creating persuasive technologies: An eight-step design process. In *Proceedings of the 4th International Conference on Persuasive Technology*, Persuasive '09, pages 44:1–44:6, New York, NY, USA, 2009. ACM.

[9] A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33. ACM, 2002.

[10] M. Gordon and P. J. Guo. Codepourri: Creating visual coding tutorials using a volunteer crowd of learners. In *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*, pages 13–21. IEEE, 2015.

[11] M. Jazayeri. The education of a software engineer. In *Proceedings of the 19th IEEE international conference on Automated software engineering*, pages 18–xxvii. IEEE Computer Society, 2004.

[12] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, ICSE '13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.

[13] M. R. Karim, Y. Yang, D. Messinger, and G. Ruhe. Learn or earn?-intelligent task recommendation for competitive crowdsourced software development. 2018.

[14] L. Layman, L. Williams, and R. S. Amant. Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 176–185. IEEE, 2007.

[15] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In M. Glinz, G. C. Murphy, and M. Pezzè, editors, *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 3–13. IEEE Computer Society, 2012.

[16] F. Linton, D. Joy, H. peter Schaefer, and A. Charron. Owl: A recommender system for organization-wide learning, 2000.

[17] C. Maltzahn. Community help: Discovering tools and locating experts in a dynamic environment. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 260–261, New York, NY, USA, 1995. ACM.

[18] C. Mendez, H. S. Pedala, Z. Steine-Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, and M. Burnett. Open source barriers to entry, revisited: A tools perspective. Technical report, Corvallis, OR: Oregon State University, Dept. of Computer Science, 2017.

[19] S. Mirhosseini and C. Parnin. 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*, pages 84–94. IEEE Press, 2017.

[20] E. Murphy-Hill. Continuous social screencasting to facilitate software tool discovery. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1317–1320, Piscataway, NJ, USA, 2012. IEEE Press.

[21] E. Murphy-Hill and A. Black. Breaking the barriers to successful refactoring. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 421–430, May 2008.

[22] E. Murphy-Hill, D. Y. Lee, G. C. Murphy, and J. McGrenere. How do users discover new tools in software development and beyond? *Computer Supported Cooperative Work (CSCW)*, 24(5):389–422, 2015.

[23] E. Murphy-Hill and G. C. Murphy. 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, pages 405–414, New York, NY, USA, 2011. ACM.

[24] D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy, and C. Swart. Reactive information foraging: An empirical investigation of theory-based recommender systems for programmers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1471–1480, New York, NY, USA, 2012. ACM.

[25] A. Rahman, A. Partho, D. Meder, and L. Williams. Which factors influence practitioners' usage of build automation tools? In *Proceedings of the 3rd International Workshop on Rapid Continuous Software Engineering*, pages 20–26. IEEE Press, 2017.

[26] L. Singer. On the diffusion of innovations: How new ideas spread, Dec. 2016.

[27] J. Smith, C. Brown, and E. Murphy-Hill. Flower: Navigating program flow in the ide. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 19–23, Oct 2017.

[28] I. C. Society, P. Bourque, and R. E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[29] C. Sunstein, R. Thaler, et al. Nudge. *The politics of libertarian paternalism. New Haven*, 2008.

[30] R. H. Thaler, C. R. Sunstein, and J. P. Balz. Choice architecture. 2014.

[31] S. Tilley, S. Huang, and T. Payne. On the challenges of adopting rots software. In *Proceedings of the 3rd International Workshop on Adoption-Centric Software Engineering*, pages 3–6, 2003.

[32] P. Viriyakattiyaporn and G. C. Murphy. 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, pages 27–41, Riverton, NJ, USA, 2010. IBM Corp.

[33] M. Weinmann, C. Schneider, and J. vom Brocke. Digital nudging. *Business & Information Systems Engineering*, 58(6):433–436, 2016.

[34] S. Xiao, J. Witschey, and E. Murphy-Hill. 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, pages 1095–1106, New York, NY, USA, 2014. ACM.