Research Statement

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

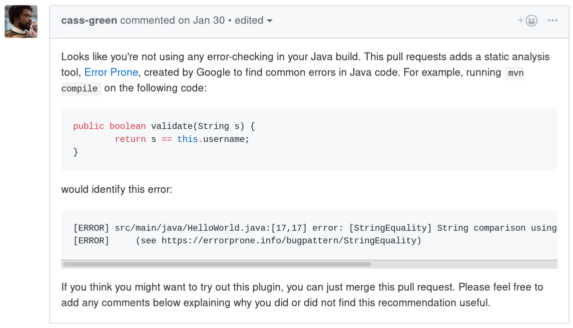
Research Overview

Decision-making plays a vital role in software engineering, and the choices developers make impact the technology we use everyday. Unfortunately, software engineers frequently make bad decisions. For example, studies show software engineers often ignore development tools, ethical programming guidelines, and other useful *developer behaviors*, which is costly for software users and producers. The motivating question of my research is: ***"How do we encourage software engineers make better decisions in their work?"***

My research interests lie in the intersection of empirical software engineering, human factors, and automation. To improve developer decision-making, my work spans disciplines to incorporate concepts from behavioral science into software engineering. Nudge theory is a framework for improving human behavior by influencing the environment surrounding decisions, or *choice architecture*, without 1) providing incentives to adopt the target behavior and 2) banning alternative choices [6]. My research introduces ***developer recommendation choice architectures***, a framework for creating automated recommendations that nudge developers towards better software engineering behaviors and practices.

My research philosophy is two-fold, to: 1) *perform empirical studies* characterizing software engineering problems; and 2) *develop tools and techniques* to overcome these problems. In my dissertation work, I explore what makes effective recommendations, construct a framework to improve automated recommendations, examine existing systems through the lens of this framework, and develop bots integrating ***developer recommendation choice architectures***. To provide evidence supporting this framework, I conducted multi-methodological studies collecting quantitative and qualitative data observing the behavior of CS students, open source software developers, and software engineers in industry. As a researcher, I will continue using this framework to observe developer actions and motivate the design of future tools for improving the productivity, decision-making, and behavior of software engineers.

Effective Developer Recommendations: To determine what makes effective recommendations to developers, I analyzed *peer interactions*, or the process of learning from colleagues during work activities, and the naive *telemarketer design*. To discover what makes peer interactions effective, we observed 13 pairs of participants completing data analysis tasks. For each session, we determined if interactions contained characteristics such as politeness, persuasiveness, and receptiveness, and if each recommendation was effective. Overall, we analyzed 142 peer interactions and found receptiveness, or *desire* and *familiarity*, was the only significant characteristic (Wilcoxon, $p = 0.0002$, $\alpha = .05$) [1]. To analyze automated recommendations, I developed the naive *telemarketer design* , a baseline approach which "calls" users to recommend and add tools to projects without the ability to customize messages or respond to users (Figure 1). We evaluated this approach in `tool-recommender-bot` on 52 GitHub repositories, and found only two recommendations (4%) were effective due to its poor *social context* and *developer workflow* [2].


cass-green commented on Jan 30 • edited

Looks like you're not using any error-checking in your Java build. This pull requests adds a static analysis tool, Error Prone, created by Google to find common errors in Java code. For example, running `mvn compile` on the following code:

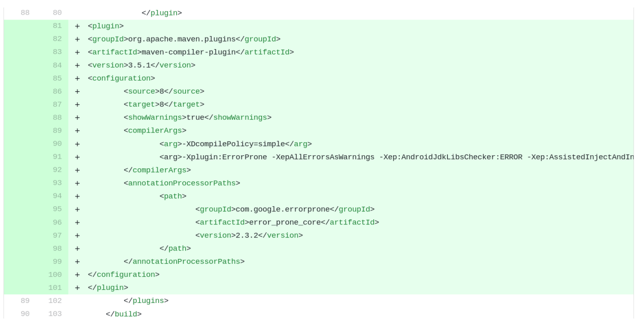
```
public boolean validate(String s) {
        return s == this.username;
}
```

would identify this error:

```
[ERROR] src/main/java/HelloWorld.java:[17,17] error: [StringEquality] String comparison using
[ERROR]     (see https://errorprone.info/bugpattern/StringEquality)
```

If you think you might want to try out this plugin, you can just merge this pull request. Please feel free to add any comments below explaining why you did or did not find this recommendation useful.

(a) naive *telemarketer design* recommendation

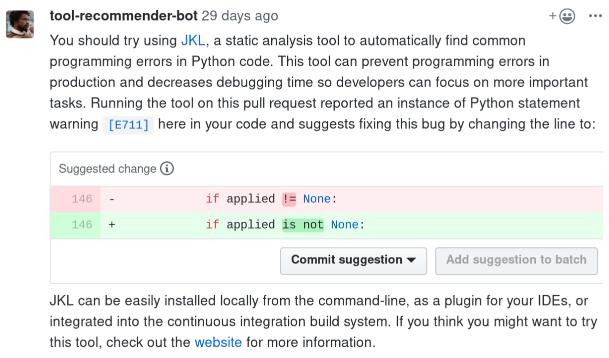


```
            </plugin>
+ <plugin>
+ <groupId>org.apache.maven.plugins</groupId>
+ <artifactId>maven-compiler-plugin</artifactId>
+ <version>3.5.1</version>
+ <configuration>
+         <source>8</source>
+         <target>8</target>
+         <showWarnings>true</showWarnings>
+         <compilerArgs>
+                 <arg>-XDcompilePolicy=simple</arg>
+                 <arg>-Xplugin:ErrorProne -XepAllErrorsAsWarnings -Xep:AndroidJdkLibsChecker:ERROR -Xep:AssistedInjectAndIn
+         </compilerArgs>
+         <annotationProcessorPaths>
+                 <path>
+                         <groupId>com.google.errorprone</groupId>
+                         <artifactId>error_prone_core</artifactId>
+                         <version>2.3.2</version>
+                 </path>
+         </annotationProcessorPaths>
+ </configuration>
+ </plugin>
        </plugins>
    </build>
```

(b) Adding tools to configuration files

Constructing a Framework: To improve automated recommendations to developers, I introduce ***developer recommendation choice architectures***, a framework for creating effective recommendations that nudge developers towards better behaviors and practices in their work. This framework consists of three principles: 1) **actionability**, or the ease with which developers can adopt the target behavior, 2) **feedback**, or the clarity and relevance of the information provided, and 3) **locality**, or the placement and timing of recommendations. In a preliminary evaluation, I surveyed professional software engineers and found 100% of participants prefer actionable recommendations over static ones [4].

Evaluating Existing Tools: To evaluate my framework, we studied recommendation styles and developer impact of GitHub *suggested changes*. This recently introduced feature allows users to recommend code improvements to pull requests, and incorporates ***developer recommendation choice architectures***. To study recommendation style, I conducted a think-aloud study where 14 professional developers interacted with static analysis tool recommendations from emails, issues, pull requests, and suggested changes (see Figure 2). We found developers significantly preferred the suggested changes recommendation (Kruskal-Wallis, $p = 0.00079$) [3].


tool-recommender-bot 29 days ago

You should try using JKL, a static analysis tool to automatically find common programming errors in Python code. This tool can prevent programming errors in production and decreases debugging time so developers can focus on more important tasks. Running the tool on this pull request reported an instance of Python statement warning `[E711]` here in your code and suggests fixing this bug by changing the line to:

Suggested change

```
146 -            if applied != None:
146 +            if applied is not None:
```

JKL can be easily installed locally from the command-line, as a plugin for your IDEs, or integrated into the continuous integration build system. If you think you might want to try this tool, check out the website for more information.

Figure 2: Suggested changes tool recommendation

To examine developer impact, I designed a study divided in two phases. Phase 1 is an empirical study analyzing suggested changes to discover types of suggestions, effectiveness compared to PR review comments, and their impact on pull-based software development. The results show most suggested changes are *non-functional*, not impacting code. Suggested changes are also more accepted and, while reviews take longer, the time to make and respond to recommendations is much faster than review comments. Furthermore, PRs with suggested changes have more commits, code churn, review comments, discussion comments, and participants in the review process. For Phase 2, we surveyed developers to qualitatively analyze feedback on the system. 98% of respondents found suggested changes at least moderately useful because of *user-driven communication* and *workflow integration* [5].

Developing Bots: To further explore the impact of ***developer recommendation choice architectures*** on behavior, I implemented `class-bot`, a system that automatically updates GitHub issues to encourage students to follow software engineering processes on programming projects. We conducted a preliminary evaluation of this bot on projects for an introductory Java course and mined GitHub repositories to observe programming behaviors on projects with and without `class-bot` notifications. Our early results show automated nudges improve code quality by increasing student grades and enhanced student productivity by increasing code churn and preventing procrastination.

Future Work

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

- ***Developer Nudges:*** One limitation of this work is the generalization of "developers" when exploring recommendations. In reality, developers are extremely diverse and have individual preferences for effective recommendations. To increase the effectiveness of automated nudges encouraging developers to adopt useful practices, I aim to study customized *developer nudges* that use different types recommendations based on user characteristics such as recent development activity, project contributions, and experience.
- ***Proactive Recommendations:*** Most of my work is *reactive*, in that recommendations occur after developers complete programming tasks. To improve the effectiveness of automated nudges, I will develop *proactive* nudges that predict developer actions and suggest useful behaviors before bad decisions are made. To accomplish this, I will apply machine learning techniques, such as collaborative filtering, to anticipate poor decisions and proactively recommend better practices to increase adoption of beneficial behaviors.
- ***Nudge Bots:*** To continue exploring effective recommendation approaches for software engineers, I aim to create an exhaustive `nudge-bot` system. This will consist of a suite of bots that: recommend various software engineering behaviors and practices; implement diverse interventions, such as GitHub Project boards, IM platforms like Slack, and automatically repairing programming mistakes; and make recommendations to developers on online programming communities such as code-hosting platforms (i.e. GitLab and BitBucket), Q&A sites (i.e. StackOverflow), and blogs and social media.

Broader Impact

My research has been published at peer-reviewed ACM and IEEE conferences and workshops. I have also presented at industry conferences such as Red Hat QECampX and DevConf. Furthermore, the tools created for this research, including `tool-recommender-bot`[1] and `class-bot`[2], are publicly available online and future systems will be made open source for developers and researchers to use, evaluate, and extend. Lastly, as a researcher from an underrepresented minority group, I plan to enable participation and increase diversity in computing research by recruiting and mentoring a diverse team of students to further explore software engineering problems.

[1]https://github.com/chbrown13/tool-recommender-bot
[2]https://github.com/chbrown13/nudge-bot

References

[1] ***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.

[2] ***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.

[3] ***Chris Brown*** and Chris Parnin. Comparing different developer behavior recommendation styles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 78–85, New York, NY, USA, 2020. Association for Computing Machinery.

[4] ***Chris Brown*** and Chris Parnin. Sorry to bother you again: Developer recommendation choice architectures for designing effective bots. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 56–60, New York, NY, USA, 2020. Association for Computing Machinery.

[5] ***Chris Brown*** and Chris Parnin. Understanding the impact of github suggested changes on recommendations between developers. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, Sacramento, CA, 2020. ACM.

[6] Richard H Thaler and Cass R Sunstein. *Nudge: Improving decisions about health, wealth, and happiness.* Penguin, 2009.

Teaching Statement

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

Teaching Approach

My instructional approach is shaped by my teaching experiences, research, and industry experiences. My teaching philosophy focuses on empowering students to pursue computer science and technology careers through *active learning*, *real-world examples*, and *diverse and inclusive learning environments.*

Active Learning: As an instructor, I will implement active learning into each class to help better understand and grasp lecture materials. To avoid teaching lectures for students to only take in information, I plan to support student learning and engagement by incorporating original, collaborative, and hands-on programming assignments and class activities. For example, a lecture about `while` loops in Python would consist of a presentation on the use of `while` loops and how they build on previous concepts covered in class, a demonstration of the `while` loop syntax, a discussion on practical examples of the concept in real life (i.e. video games), and a group activity for students to work together to implement Python programs that incorporate `while` loops.

Real-World Examples: The best way to prepare for a computer science students for industry careers is experience. Thus, as a professor I will attempt to simulate authentic programming experiences into classes by incorporating real-world software engineering tools and practices into courses. For example, I will provide realistic projects for students to practice skills necessary to develop and maintain production-quality software, grade programs using realistic code quality metrics such as passing unit tests and static analysis tool warnings, and encourage students to participate in industry practices such as pair programming, debugging, and code reviews. Additionally, I envision integrating my personal background and skills learned from my industry experiences into course lectures and materials. Furthermore, I will include relevant software engineering research findings into courses to increase awareness of cutting edge tools and practices among students and future developers, including my own work on improving developer behavior, decision-making, and productivity.

Diversity and Inclusion: Lastly, I will create open classroom environments that foster equal opportunities for all students to learn no matter their experience or background. As a course instructor, I believe it is vital to listen and support the needs of students in the classroom and one-on-one settings. This includes creating accessible course materials, accommodating different learning styles and preferences in lectures, and making course requirements and expectations clear. Additionally, I aim to provide multifaceted active learning opportunities for computer science majors and non-majors who may not want to pursue predominately coding careers by incorporating interdisciplinary materials. Finally, I will ensure all students have the opportunity to excel in Computer Science courses I teach independent of their race, gender, age, sexual orientation, programming experience, etc.

Teaching Experience

Instructor of Record: I was the primary instructor for the introductory Java programming course, *Introduction to Computing: Java* (CSC116), at NC State for the 10-week semester during Summer 2020. As the instructor of record, I was responsible for preparing lecture materials, delivering lectures, holding office hours, assigning and grading projects, labs, and homework, managing teaching assistants and graders, and more for a diverse class of students from various majors and backgrounds. Due to COVID-19, the class was taught remotely over Zoom instead of in-person. To incorporate active learning and increase student engagement virtually, I instituted a flipped classroom by prerecording lectures for students to watch asynchronously then integrating programming lab activities based on the lecture content during the designated lecture time. This was a valuable opportunity to motivate my teaching approach and gain experience as an instructor teaching Computer Science concepts to students.

Teaching and Communication Certificate: I am currently enrolled in the North Carolina State University Graduate School Teaching and Communication Certificate program to gain additional experience and training for effective communication skills. The Teaching and Communication Certificate requires completing 100 hours of approved activities and creating a professional development portfolio. Through this program, I have taken useful courses to improve my teaching approach with a focus on enhancing inclusivity in higher education such as *Accessibility in the Classroom* and *Teaching about Identities, Diversity, and Equity* (ECI509) to gain knowledge and learn techniques for creating accessible and inclusive content and materials for students.

Teaching Assistant: I was a Teaching Assistant (TA) for two undergraduate computer science courses as a Ph.D. student at NC State: *Software Engineering* (CSC326) in Fall 2015 and *Programming Concepts - Java* (CSC216) in Spring 2016. For CSC326, I was expected to teach two lab sections emphasizing concepts taught in lectures, act as the Scrum Master for teams during the final project, grade programming assignments and exams, attend lectures, answer student inquiries via email and Piazza, and hold office hours. For CSC216, I aided students during in-lecture activities, graded assignments and exams, and held office hours. These TA opportunities allowed me to gain experience interacting with students, performing instructional tasks, and guiding active learning activities.

Research: My research interests also motivate my teaching approach and span topics in Computer Science education. For an independent study, I developed a grading algorithm that incorporates automated program repair concepts to improve student feedback on introductory programming projects. Compared to a baseline approach, my algorithm increased student grades and only took slightly longer to run on a benchmark suite. For my dissertation work, I implemented `class-bot`, a system to improve student behavior by automatically updating GitHub issues on repos to notify students of their progress following software engineering processes. We found `class-bot` improved both project quality and student productivity. Additionally, I am currently working on a bot to increase student engagement and encourage consistent contributions and collaboration on team projects. As an instructor, I will continue to explore automating instructional activities and improving student programming behaviors to prepare them for real-world technology careers.

Tutoring and Service: I have also gained teaching experience through outreach focused on diversity and inclusion. In Summer 2019, I worked for The Coding School *codeConnects* program to increase participation of underrepresented groups in STEM. Through this mentorship and tutoring program, I spent 10 weeks teaching Python to a high school student online. I have also participated in many service opportunities focused on teaching Computer Science. With the NC State Minority Engineering Graduate Student Association (MEGSA), I volunteered at an INTech Mini-Camp to teach HTML and website design to African American middle school girls. Additionally, through the NC State Students and Technology in Academia, Research and Service (STARS) Computing Corps I volunteered to help teach Scratch, a 2D block-based programming language, to local underserved middle school students. These tutoring experiences and volunteer opportunities helped me gain experience teaching computing concepts to diverse groups of students.

Example Courses

Based on my experiences and interests, I believe I am equipped to teach undergraduate or graduate level *software engineering* courses to help students gain the necessary skills to build and maintain software systems. I can also teach *introductory programming* classes to teach basic coding concepts to students with little or no programming experience. I will be most effective teaching these types of courses in Python or Java, but I am also able to complete them in other programming languages. Below is a sample list of additional courses I would be excited to teach based on my interests, experiences, and current topics within the field:

- ***DevOps:*** A graduate or advanced undergraduate level course that focuses on providing an overview of DevOps concepts. In the course, students would learn about development and operations industry practices and tools for continuous integration, deployment, project builds, and more. The course would involve learning about these processes as well as completing a final project to automate software development tasks.
- ***Open Source Software:*** Another software engineering undergraduate or graduate level course focusing on contributing to open source software (OSS). This class would go over topics such as the importance of OSS, licensing, open data, and civic tech. Students would also be required to contribute to real-world open source repositories, such as projects in the Humanitarian Free and Open Source Software (HFOSS) Project[1] that produce software to solve real-world problems and benefit communities, throughout the semester.
- ***Software Engineering Research Seminar:*** An undergraduate or graduate level course that explores current topics in software engineering and computer science research. This discussion-based course would involve students reading publications, learning research practices, and designing a study to complete for a publication-quality paper.

Overall, I believe I can excel as a computer science educator for students by incorporating active learning into classrooms, helping students gain real-world knowledge and skills through course lectures, assignments, and projects, and creating engaging and inclusive classroom environments. With my teaching approach, prior experiences, and research interests, I will strive to provide relevant materials to support students to succeed beyond the course and to prepare them for future careers as computer scientists.

[1]http://www.hfoss.org/