Research Statement

Brittany Johnson (bjohnson@cs.umass.edu)

My research investigates software development practices. More specifically, I study if and how existing tools and techniques support developers' ability to produce high quality, ethically-sound software and develop new tools to evaluate hypotheses regarding developer needs during the software development process. The goal of my research is to create and disseminate empirically validated theories and techniques that can improve developer tools and processes for a positive and practical effect on their productivity and quality of the resulting software product. I regularly publish in top conferences and journals in software engineering, including ICSE [7, 12], FSE [8, 9, 16], and TSE [11, 17].

Improving Software Ethics with Empirically Validated Tools & Processes

Advances in modern technology have provided developers with the means for building software that makes smarter, more objective decisions and automating more tasks. Because of these advances, software is increasingly being used to replace human intervention in our day to day lives with the goal of improving quality of life; from healthcare, where software is being used to diagnose and treat medical conditions [6], to the criminal justice system, where some states are using software to determine criminal sentencing and bail [2].

While machine learning allows software to automate more decisions, and has the potential to decrease bias in decision-making, it also increases the likelihood that software will behave in unexpected and undesired ways that can have a negative impact on quality of life. This potential for negative societal impact presents an ethical challenge regarding how software should work and, more importantly, the role developers play in minimizing the negative impact these software systems can have.

Now, more than ever, it is important for us to understand the tools and processes developers use to build software so that we can ensure they are equipped to effectively build ethically-sound, high quality software. Research suggests that we can support developers while completing development tasks by reducing developer effort (e.g., providing tools that automate manual tasks) [3]. Therefore, for developers to make ethical considerations like fairness while still being productive, we need to provide tools, techniques, and resources that effectively support this additional step.

The goal of my research is to help developers efficiently and effectively produce high quality, ethically-sound software by evaluating and improving the tools and processes they use in their day to day work and development environments. My research vision is for developers to be empowered with tools and practices that effectively support their ability to produce high quality, ethically-sound software.

1 Research Contributions

My vision is to provide validated tools, techniques, and practices that support ethical software development. Before I can develop interventions developers want to use, it is necessary to understand the kinds of tools developers use, how and why they use them, and barriers to use that could affect the adoption or use of new technologies. To improve this understanding, my prior work investigated developer tool use and discovered that one of the primary reasons developers do not use existing tools is difficulty interpreting tool output and that this difficulty stems from various kinds of mismatches between how tools communicate information and developers' expectations. Using these findings, I have begun to explore tooling we can provide to developers that support the completion of development tasks, like debugging, and the evaluation of software for ethical concerns, like fairness.

1.1 Empirical Studies on Developer Tool Use

Foundational to my proposed research are the empirical studies I conducted to explore barriers to developer tool use and productivity [7, 9, 11]. The first asked why developers do not use their tools [7]. Based on this interactive interview study of 20 professional developers, one reason developer don't use tools is difficulty interpreting tool notifications. This finding informs our understanding of why developers do (and do not) use tools.

Based on these findings, my second foundational work explored why developers encounter challenges when interpreting tool output [9]. I conducted sessions with 26 developers, all with different backgrounds and years of programming experience and used think-aloud and observation protocols to collect data on the challenges they encounter when understanding tool output. Based on the findings, I proposed a *tool (mis) communication theory*, which states that the challenges developers encounter stem from gaps and mismatches between what developers expect based on their knowledge and experiences and how tools communicate.

1.2 Removing Barriers to Tool Use

Using empirical foundations from my prior work, and work done by others, I have explored solutions to the tool underuse and miscommunication problems. Though tools often help developers figure out the existence of a problem, figuring out why the tool is complaining and how to fix the defect is often left to the developer [7]. To address this issue in testing tools like JUnit, which currently only help find defects, I developed and evaluated a new, complementary testing technique called *Causal Testing* [12]. Causal Testing is a novel method for identifying root causes of failing executions based on the theory of counterfactual causality. However, instead of using observational data, my approach takes a manipulationist approach to causal inference [18], modifying and executing tests to observe causal relationships and derive causal claims about the defects' root causes.

Causal Testing generates similar passing and failing tests along with their execution traces, which allows developers to compare passing and failing inputs and execution paths. To evaluate the usefulness of Causal Testing for debugging, I conducted a user study with 37 developers where I asked them to debug six failing tests using JUnit and Holmes, a proof-of-concept Causal Testing implementation. The results of this evaluation suggests Causal Testing is complementary to tools like JUnit and can improve developers' ability to determine the cause of a failing test, thereby improving their ability to remove the defect.

Along with being useful for debugging, Causal Testing should be able to scale to the different kinds of tests developers write and bugs they find. Therefore, along with a user study, I conducted an applicability evaluation on the Defects4J benchmark [14] where I manually categorized 330 defects based on whether Causal Testing could be applied and the potential for information provided to be useful for debugging. Findings from this evaluation suggested Causal Testing could be applied to 71% of real-world defects in the Defects4J benchmark, and for 77% of those, it could help developers identify the root cause of the defect.

Another solution I have explored for improving existing tools is the ability to build and use developer knowledge models, trained using developer code contributions, to adapt the way tools communicate to the developer using it [8, 10]. To evaluate the feasibility of predicting developer knowledge, I developed an approach that uses *concept-specific code* from public repositories to predict developer knowledge of a given programming concept [8]. Concept-specific code maps directly to a programming concept, such as a generic type declarations which map to the concept of generics. My approach automatically collects developer code contributions and performs data reduction based on data features and heuristics from existing literature, such as how recently the code was written [4]. Predictions of developer concept knowledge based on concept-specific code are more accurate than those based on all code ever written and yield accuracy between 47% and 78%.

1.3 Tooling to Support Building Fair Software

Just as practice has to keep up with changes in society and technology, as should research. Therefore, an important problem I will explore is providing a better understanding of and support for ethical software engineering practices. More specifically, my previous work has focused on tooling to help developers evaluate software systems that use machine learning models for bias [1, 13].

Fairness is a new concern for developers added to the list of other concerns when developing software, along with things like performance and security. While fairness is an important consideration, a fair and inaccurate model can have similar negative impacts as an accurate yet unfair model. To help developers reason about trade-offs when considering both fairness and other metrics of quality, my prior work evaluated a novel model exploration tool called fairkit-learn [13]. Using fairkit-learn, developers can attempt to reason about fairness and other quality metrics simultaneously.

To evaluate fairkit-learn in comparison with the state-of-the-art in model training and evaluation, I ran a within-subjects user study with 54 students in an advanced software engineering course. Findings from this study suggest that developers can use fairkit-learn to produce more fair models than when using competing toolkits scikit-learn[1] or AI Fairness 360 [19]. Data from this study also provided insights into how developers reason about fairness in machine learning models when using traditional tools, like scikit-learn, that do not include functionality for evaluating for ethical concerns like fairness. More specifically, this study found that developers may have different, sometimes contradictory, ways of reasoning about fairness when asked to do so, often using accuracy as a proxy for fairness, despite clear evidence that those metrics are often at odds.

Another solution I contributed to with regards to software fairness is a tool called Themis [1]. Themis is an automated test suite generator that provides functionality for detecting and measuring discrimination based on sensitive inputs. Using Themis, developers can generate tests that can help expose biased, or unfair, behavior with regard to protected attributes such as gender or race. Previous work found that Themis' technique is effective in a controlled setting [5]; as I discuss in my research agenda, I will use my expertise in empirical studies to explore how well tools like Themis work in practice and improvements needed to make fairness testing scalable.

2 Research Agenda

My research agenda further develops and expands on my previous contributions, with the goal of providing tools and techniques that support developers while making ethical considerations, like fairness, during the software development process. Research I plan to work and build on includes the following.

[1]`http://scikit-learn.org/stable/`

2.1 The Role of Ethics in Software Development

Advances in technology, such as the increasing ubiquity of machine learning in software, have led to advances in how software is integrated into everyday life. However, with great power comes great responsibility. Now that software can automate more decisions, the potential for software to have a negative or unfavorable impact on society is much greater. While my prior work has focused on bias in software [1, 13], fairness is just one of the many potential ethical concerns regarding software development and use. Furthermore, despite the progress in ethical tooling and processes, it is still unclear which (if any) of these are used in practice.

Building on my prior work, and work done by others in this area [15, 19], I will investigate the role developers can and should play in building and deploying software that is designed, implemented, and maintained with explicit and intentional considerations regarding the social and legal effects the software will have on its diverse set of users. More specifically, my research will explore 1) existing ethical software development tools and practices, 2) developer perceptions of ethical concerns related to software production, 3) gaps in existing processes that hinder ethical software practices, and 4) solutions that help fill those gaps. One direction I plan to explore in the near future is the types of ethical defects developers may encounter and the ability for tools like Themis and techniques like Causal Testing to help debug these ethical defects.

2.2 Development Practice Effectiveness

It is important to evaluate how developers build software to determine the ability for a given intervention, such as a tool or process change, to do what it was designed to do. As I have shown with my prior work [7, 9, 11, 12], empirical studies are useful for better understanding a given phenomenon and can provide foundations for improving developer tools and processes. However, despite numerous studies on how and why developers adopt and use tools and processes, it is still unclear which are effective (and which are not), and more importantly how we can define tool effectiveness so software teams can know the difference.

I will conduct empirical studies that attempt to understand and define what it means for a software development practice to be effective, starting with existing practices. This includes, but is not limited to, how developers and other software stakeholders define effectiveness, the ability for a given practice to foster or hinder developer productivity, and the relationship between developer and end-user perceptions of software practices and the resulting product. These evaluations will validate, or disprove, existing hypotheses, potentially providing new directions to explore, and provide new tools, techniques, and insights for improvements and future evaluations.

As I have outlined, there are numerous directions my research can and will go. Given the interdisciplinary nature of my research, I will continue to pursue collaborations with students, other academic researchers, and industry to address these research challenges. I strongly believe that through a deeper understanding of developer practices and needs when developing software, we can improve practice and the resulting software products.

References

[1] Angell, R., **Johnson, B.**, Brun, Y. & Meliou, A., (October 2018). *Themis: Automatically testing software for discrimination.* In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 871-875). ACM.

[2] Angwin, J., Larson, J., Mattu, S., & Kirchner, L. (2016).*Machine Bias.* ProPublica. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[3] Bruckhaus, T., Madhavii, N. H., Janssen, I., & Henshaw, J. (1996). *The impact of tools on software productivity.* IEEE Software, 13(5), 29-38.

[4] Fritz, T., Ou, J., Murphy, G. C., & Murphy-Hill, E. (2010, May). *A degree-of-knowledge model to capture source code familiarity.* In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1(pp. 385-394). ACM.

[5] Galhotra, S., Brun, Y., & Meliou, A. (August 2017). *Fairness testing: testing software for discrimination.* In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (pp. 498-510). ACM.

[6] Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., & Wang, Y. (2017). *Artificial intelligence in healthcare: past, present and future.* Stroke and vascular neurology. 2(4), 230-243.

[7] **Johnson, B.**, Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013, May). *Why don't software developers use static analysis tools to find bugs?.* In 2013 35th International Conference on Software Engineering (ICSE) (pp. 672-681). IEEE.

[8] **Johnson, B.**, Pandita, R., Murphy-Hill, E., & Heckman, S. (2015, August). *Bespoke tools: adapted to the concepts developers know.* In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 878-881). ACM.

[9] **Johnson, B.**, Pandita, R., Smith, J., Ford, D., Elder, S., Murphy-Hill, E., & Sadowski, C. *A Cross-Tool Communication Study on Program Analysis Tool Notifications.* In 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE).

[10] **Johnson, B. I.** (2017). *A Tool (Mis) communication Theory and Adaptive Approach for Supporting Developer Tool Use* . North Carolina State University.

[11] **Johnson, B.**, Zimmermann, T., & Bird, C. (February 2019.)*The Effect of Work Environments on Productivity and Satisfaction of Software Engineers.* IEEE Transactions on Software Engineering.

[12] **Johnson, B.**, Brun, Y., & Meliou, A. (2020). *Causal Testing: Understanding Defects' Root Causes.* To appear in the Proceedings of the 2020 International Conference on Software Engineering.

[13] **Johnson, B.**,Bartola, J., Angell, R., Keith, K., Witty, S., Giguere, S., & Brun, Y. (2019). *Fairkit, Fairkit, on the Wall, Who's the Fairest of Them All? Supporting Data Scientists in Training Fair Models.* In submission.

[14] Just, R., Jalali, D., & Ernst, M. D. (2014). *Defects4J: A database of existing faults to enable controlled testing studies for Java programs.* In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 437-440). ACM.

[15] McNamara, A., Smith, J. & Murphy-Hill, E. (October 2018). *Does ACM's code of ethics change ethical decision making in software development?.* In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 729-733). ACM.

[16] Smith, J., **Johnson, B.**, Murphy-Hill, E., Chu, B., & Lipford, H. R. (2015, August). *Questions developers ask while diagnosing potential security vulnerabilities with static analysis.* In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 248-259). ACM.

[17] Smith, J., **Johnson, B.**, Murphy-Hill, E., Chu, B. T., & Richter, H. (2018). *How developers diagnose potential security vulnerabilities with a static analysis tool.* IEEE Transactions on Software Engineering.

[18] Woodward, J. (2005.) *Making things happen: A theory of causal explanation.* Oxford University Press.

[19] Bellamy, R. K., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., & Nagar, S. (2019). *AI Fairness 360: An Extensible Toolkit for Detecting and Mitigating Algorithmic Bias.* IBM Journal of Research and Development.