RESEARCH STATEMENT

Michael Hilton (hiltonm@eecs.oregonstate.edu)

My research is motivated by the decade I spent as a software developer. In my experience, having a good process is crucial for a developer to be successful. However, improving one's process is very difficult. The first step in improving one's process is understanding. There are many ways for a developer to evaluate the product of their work (e.g., the code they write), however, there are not many ways to evaluate the process of their work (e.g., *how* they write code). It is only through understanding developers' processes that we can develop tools that will be most effective at addressing the core needs of developers.

I believe that it is important for developers to better understand their own development, but also for researchers to better understand how software is created, so that they can contribute to process improvement. My research focuses on increasing understanding of software development processes, specifically focusing on how developers use the process of Continuous Integration(**ASE** [4]). My research also helps developers better understand their own development process (**XP** [3]). Another aspect of my research includes taking established processes such as refactoring, and using them in novel ways, to help developers. For example, developing refactorings to automatically add cloud data types to mobile applications (**MOBILESoft** [1]).

I believe that another reason to study processes is to improve computing education. If we understand what a good development process looks like, then we can teach good processes to the next generation of developers. My education research(**ITiCSE [2], ICSE Education Track [5]**) uses Test-Driven Learning as a process to improve outcomes for beginning students.

I also believe that collaboration is an important part of research. During my PhD, I have actively pursued collaborations. I have sought out senior collaborators to learn from and also made a concerted effort to collaborate with undergraduates and junior students. I have collaborated with researchers from Carnegie Mellon University, Iowa State University, Microsoft Research, University of Colorado Boulder and University of Illinois at Urbana-Champaign. These fruitful collaborations have led to advances in improved code recommendation(**FSE Distinguished Paper Award** [6]), and exploring the foundations of structured editing(**POPL** [7]). I also successfully collaborated with undergraduates. These undergraduates participated in NSF-funded Research Experience for Undergraduates (REU) at Oregon State University, as well as independent study participants from UIUC. It has been rewarding to help them learn about research, and develop into productive members of a research team. I have found partnering with undergraduates to be a valuable experience, and one that I plan on continuing in the future.

Research Experience

Continuous Integration

Continuous Integration (CI) is a process that involves merging, building, and testing code continuously, or at least very frequently. It has been widely adopted by the software development industry. However, little is known about how CI is used in practice. Also, while proponents of CI make claims about its benefits, those claims had not been substantiated. Without knowing how it is used, and if the claimed benefits exist, developers are forced to make decisions based on their intuition, and researchers ignore a widely used practice in industry. In our research(**ASE** [4]), we performed a large study to examine the usage of CI in Open Source projects. We collected data from over 34,000 projects on GitHub, and for a subset of those, we collected their entire CI history. Our research shows that 40% of of 34,000 of the most popular projects on GitHub use CI. Among the top 500 projects, 70% of the projects use CI. One of the most common claims about CI is that it helps developers release more often. In our work, we show that projects which use CI release twice as fast as projects who do not use CI. We also looked at CI usage by language. We found that projects which use dynamically typed languages are more likely to use CI than statically typed languages. While further work is needed to determine exactly why this is, our intuition is that developers who are using dynamically typed languages use CI, and specifically unit tests, to mitigate the loss of a type system, and the safety type systems provide. This study of Open Source was very revealing, but we still didn't know how CI was being used by proprietary developers.

In our most recent work, currently under review, we performed a qualitative study of developers who use CI to determine their needs and wishes. We interviewed 16 developers using semi-structured interviews. We used what we learned from these interviews to create a survey which we deployed to over 500 developers to obtain a diverse set of opinions about CI. We asked developers questions about their motivations for using CI, how they experienced using CI, barriers to adoption, problems they encounter, and unmet needs. The most common motivation we encountered for using CI is that developers feel that CI helps them catch errors earlier, and they worry less about breaking the build. Developers also told us that they felt that CI leads to higher quality tests, and developers who use CI give more value to

tests. Developers also told us the most common problems they encounter when using CI are difficulty troubleshooting CI build failures, and overly long build times.

In our recent work, we also replicated one of the questions we asked of open source developers, but targeted developers of proprietary code. We found that developers of proprietary code often encounter political hurdles to implementing CI. By contrast, Open Source developers say the lack of knowledge of CI by the other developers on their project is the primary reason that they don't use CI. Moreover, while many Open Source projects are able to run their CI quite quickly, we found that long running build times was a common problem experienced by proprietary developers. Many developers of proprietary code described using test selection techniques to keep their running times manageable. Another common problem experienced by developers was flaky, or non-deterministic tests.

This research was done with undergraduate researchers from UIUC. I was responsible for managing the student researchers and enabling their participation throughout the project. Not only were they able to learn about data collection and analysis, but they were able to become experts in how the process of CI is implemented.

Software Development Process Understanding Through Visualization

The first step to improve a process is to understand it. We cannot expect developers to improve their processes if we do not give them the tools they need to understand and analyze their processes. One process for developing software is Test-Driven Development (TDD). TDD is a process where a developer writes just enough tests to get the test to fail, then writes enough code to get the test to pass, and then refactor if needed. To help developers understand their development process, we developed TDDViz(**XP** [3]). TDDViz helps developers by allowing them to visualize their process. TDDViz infers if the developer is using TDD, and if so, which phase they are in. This allows a user to review their development process, and *identify* how well their development process follows TDD. The visualization allows them to drill down and *comprehend* when and why specific parts of the code were changed. Finally, having this visualization allows developers to *compare* their development process to that of other developers. Three of the authors on this paper were undergrad researchers, who I mentored over the summer.

Mobile Refactoring for Cloud

Mobile cloud computing can greatly enrich the capabilities of mobile devices. Cloud computing can enable seamless transition between devices, multiuser support, and automatic backup. However, implementing cloud functionality for mobile apps is a difficult and time-consuming process, and requires a large amount of background work. We developed CLOUDIFYER (**MOBILESoft** [1]), to automatically refactor local data types to cloud data types. We built CLOUDIFYER after converting four apps to use cloud data as a part of a formative study. To evaluate CLOUDIFYER, we evaluate it with a corpus of 123 real world apps, and we showed that CLOUDIFYER is widely applicable, and it is accurate.

Education Research

Another important motivation for understanding processes is being able to teach others. In our works (**ITiCSE [2], ICSE Education Track [5]**), we focus on helping beginning students learn basic programming concepts, by following the Test-Driven Learning (TDL) process. TDL is a process where the instruction is driven by the students first writing tests, and then writing code to make the tests pass. We implemented this in a CS0 class. When using this approach, despite weaker pre-study assessments, students instructed with Test-Driven Learning performed up to twelve percent better on final assessments than the students who used traditional labs.

API Code Recommendation

Learning and remembering APIs is a challenge for developers. Not only are there many APIs that a developer may use in a single piece of code, these APIs can change, leading to developers' knowledge of them getting stale. Having to navigate a long list of API calls can be overwhelming to developers. In our work(**FSE Distinguished Paper Award** [6]), we use the insight that code changes themselves are natural, and building on previous work, we use code context, but also build statistical learning from code changes to improve our recommendations. Using this insight, we achieve 30-160% improvement in top-1 accuracy over previous approaches. This work was a collaboration with Iowa State University. Though I was not the first author, I performed the evaluation, as well as mentored the two authors who were undergraduates on a summer REU.

Structure Editor

Structure editors allow programmers to directly edit the program tree. Because each edit is comprised of atomic tree edits, this ensures that syntactic errors can never occur in programs. In our work(**POPL** [7]), we introduce Hazelnut,

a structure editor based on a small bidirectionally typed lambda calculus extended with holes and a cursor (la Huets zipper). Hazelnut goes a step further than traditional structure editors, and gives static meaning to intermediate edit states and the transitions between them. We accomplish this by defining a hole type, which can hold terms that would otherwise be type inconsistent. This approach can open the door for researchers to understand the process that developers follow when writing code in ways that were not previously possible. This collaboration arose when I met the first author, Cyrus Omar, at a conference. My co-authors on this paper are Programming Language (PL) researchers, and working with them allowed me to gain a better understanding of the PL community, while also contributing to meaningful research.

Future Work

For my future work, in addition to continuing my current collaborations, I plan on focusing on two areas: Continuous Integration, and Education Research.

Continuous Integration

Without data from real developers, researchers cannot be sure how practical their results will be. While there is much that can be learned from studying repositories, there is a lot of information that is not captured in a repository. For example, when working with repositories, there is no way to know which tests were run, and what the results of the tests were at that point in time. However, this data is available from CI services.

CI servers compile the code, run tests, and record the output. Many CI servers also collect metrics including test coverage. Since CI servers operate in conjunction with source control, they also contain the code that was run at that specific point in time. Moreover, many systems also include features such as issue tracking, where failures in compilations and tests can be traced to issues which can then be traced to code patches. This data offers many new opportunities for research that were previously unavailable. One initial line of work could be validating current test research, with new real-world data from a large number of projects. This could provide significant insight into how certain testing approaches work in various different situations.

Moreover, what information that the CI system doesn't provide, could be captured by adding minor instrumentation at the CI server. The CI already has access to the entire code, and is building and testing it. By injecting instrumentation into the CI server, I can study almost any aspect of the development process. For example, with some minor instrumentation of the IDE, the CI server can perform analysis on how developers change the code, how code completion was used to write that code, and much more.

To implement this research plan, I will involve undergraduate students in the research. This work involves data collection and analysis, learning CI systems and processes, and creating and conducting research with human subjects. These are all feasible for students to help with. Doing research will help students develop their critical thinking skills, that will help them succeed for the rest of their lives. Researching CI will also provide them with exposure to many real-world projects, which will help them distinguish good practices and high quality code from poor practices and inferior code. Moreover, for some students, this will encourage them to pursue a career in research.

Education Research

Another focus area for my future research is Computing Education.

Computing has the power to change lives. For many students, computing can be the path to changing their lives. However, our computing education system currently does not support all backgrounds and learning styles equally. This imbalance has led to large demographic groups being significantly underrepresented in computing education.

By focusing on how to better support under-represented groups, it is quite likely that we will find that our results generalize to the point that they provide better results for many students in groups that are not underrepresented. This will lead to better outcomes for all students.

One example of the education research I plan on doing is creating a curriculum around CI. Before beginning this line of research, I attempted to introduce CI into the software engineering curriculum I was teaching. I found that there was very little research on how to use CI properly, let alone on how to teach it. Once I have successfully developed ways to teach CI, I plan on publishing this research in venues such as SIGCSE, or ICSE Software Engineering Education and Training.

My long-term goal is to become a leader in helping to improve the understanding and instruction of software processes in core SE classes across academia.

References

[1] M. Hilton, A. Christi, D. Dig, M. Moskal, S. Burckhardt, and N. Tillmann. Refactoring local to cloud data types for mobile apps. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, pages 83–92. ACM, 2014.

[2] M. Hilton and D. S. Janzen. On teaching arrays with test-driven learning in webide. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 93–98. ACM, 2012.

[3] M. Hilton, N. Nelson, H. McDonald, S. McDonald, R. Metoyer, and D. Dig. TDDViz: Using Software Changes to Understand Conformance to Test Driven Development. pages 53–65. Proceedings of Agile Processes, in Software Engineering, and Extreme Programming: 17th International Conference (XP'16), 2016.

[4] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 426–437, 2016.

[5] D. S. Janzen, J. Clements, and M. Hilton. An evaluation of interactive test-driven labs with webide in cs0. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1090–1098. IEEE Press, 2013.

[6] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig. Api code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 511–522. ACM, 2016.

[7] C. Omar, I. Voysey, M. Hilton, J. Aldrich, and M. A. Hammer. Hazelnut: A bidirectionally typed structure editor calculus. *The 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2017. to appear.