

# Research Statement

Jaime Spacco

## 1 The Need for Computer Science Education Research

Recognizing the need for a highly-qualified workforce for the future, industry mainstays such as Microsoft and Google have recently funded initiatives to improve the state of the art in computer science education, while the National Science Foundation has made funding innovations for computer science and engineering curricula a priority. These trends illustrate the primary importance of understanding and improving how to teach computer science.

Many educators hold strong opinions about the “right” way to teach introductory students to program, which leads to lively and spirited debates. However, how novices learn to program is actually poorly understood. Many conclusions are based on “folk-wisdom” and anecdotal evidence rather than rigorous, repeatable studies. I believe a major factor behind the dearth of solid studies is that nobody has collected enough good data on the behaviors and attitudes of introductory computer scientists to enable these sorts of studies.

My research so far sheds light on *what* students are doing when learning to program by collecting an extensive dataset on programming practices. I’ve also begun studying *how* novices program by studying how well they use the tools available to them, such as debuggers and unit testing frameworks. The major open question I’m pursuing is *why* students are doing what they’re doing. Tackling this last question will require collaboration with researchers in cognitive science, education, and psychology.

## 2 The Marmoset Project

Learning to program can be a very frustrating experience for many students. To better understand and improve the experience of novice programmers, I’ve built Marmoset, an integrated submission, grading and data collection system for programming courses. Marmoset is similar to other automated testing systems in that students can upload their submissions to Marmoset, which runs each against instructor-provided test cases and stores the results.

Marmoset also provides a unique pedagogical innovation called *release testing*. A release test gives students controlled feedback from the instructor’s private grading tests, but does so using a token-based system rather than simply giving students all of the grading tests (and therefore encouraging students to code to the tests). More about release testing can be found in my teaching statement.

Marmoset is production-quality software that has been used at the University of Maryland since the Fall 2004 semester. During the current semester (Fall 2006), Marmoset is being used at Maryland in 9 programming courses by over 700 students and two dozen instructors and TAs. In addition, the University of Waterloo in Canada and Colgate University have adopted Marmoset for the Fall 2006 semester, and York College plans to adopt Marmoset in the Spring

2007 semester. Currently Marmoset builds and automatically tests programming assignments written in C/C++, Java, Ruby, Python and OCaml, with 24-hour-a-day reliability.

## 2.1 Data Collection with Marmoset

To better understand *what* students are doing when learning to program, Marmoset provides a light-weight plugin for students' Integrated Development Environments (IDEs) that captures snapshots of students files to a central repository after every save operation. The resulting dataset is enormous—over 150,000 unique compilable snapshots over four consecutive semesters of CS-2 at the University of Maryland—and provides a wealth of information for further study. In addition, these snapshots are extremely fine-grained: over 70% of snapshots change *four or fewer lines of code*.

The Marmoset dataset represents the most comprehensive collection of data on novice programming behaviors that currently available. I have used this dataset for a variety of studies:

- **Mining novice mistake patterns:** By making queries such as “what are all the one-line changes that caused more than three test cases to start passing/failing?” of the Marmoset dataset, we can find changes that are candidates for likely mistake patterns. Using this approach, we've identified a number of common novice mistake patterns, and have added detectors for these patterns to FindBugs, a popular open-source static checker for Java code. In addition to mining new bug patterns, we've also used the Marmoset dataset to evaluate the precision and recall of many existing FindBugs detectors, with great success.
- **Educational Data Mining:** I've collaborated with a colleague in the educational data mining community on using clustering algorithms on test case outcomes with the purpose of identifying related unit tests used to grade student programs. This work is in its infancy, but there is tremendous potential for future work in this area. For example, performing clustering on both test cases and exam questions could help identify how underlying concepts taught in class are used in programming assignments. Ultimately, all of this information can be used by instructors test student programs more effectively.
- **Test-Driven Development (TDD):** Test-Driven Development encourages programmers to write test cases as they develop the code, rather than writing the code first and then testing it later. TDD has many advantages, such as helping find bugs earlier in the development cycle. At the University of Maryland, we have taken advantage of Marmoset's integrated support for student-written test cases and code coverage to experiment with a variety of incentives to encourage student testing. We used the snapshots collected by Marmoset to isolate when students added the majority of their unit tests. We found that simply telling students that their test suites would be graded caused students to write more tests, but many students wrote the majority of their test cases *after* they had finished coding the project. Other incentives, such as giving students less feedback for poor test suites and better feedback for better test suites, does motivate the desired behavior. This work with TDD is important because it can help determine whether testing should be formally taught as part of the CS curriculum or left as a useful skill students pick up outside the classroom as they progress through the major.

## 3 Future Research Directions

### 3.1 Expanding Marmoset

Marmoset collects a large amount of data that describes *what* students do when they program. We can study *how* students program by examining how they interact with their programming environment. This is an important step in identifying productive and unproductive behaviors and helpful new technologies. Towards this end, I've modified Marmoset to track when students use some of the advanced features of Eclipse, such as the debugger.

In the Spring 2007 semester I will be participating in a multi-institution study that will use data collected by the Marmoset infrastructure to examine the effect using a production-level debugger has on CS-1 students. There has not been a controlled study that addresses this question, and it's a very important question. Skills that may be helpful later in the major, such as using a debugger, are not usually taught in the CS curriculum; instead, students are expected to pick up these skills along the way through the major. If our study shows that use of the debugger makes a substantial improvement to students' programming abilities, then perhaps using a debugger should be the focus of a lecture or a lab session.

### 3.2 Collaboration with learning sciences

The Marmoset dataset helps us answer what students are doing, and how they are doing it, but does not give us much insight into *why*. Answering this question will require research that focuses more closely on students' attitudes, motivations, levels of understanding, and so on, and can be collected through a variety of means—for example, surveys, “think-aloud” studies, or controlled laboratory sessions. There is a tremendous opportunity here for collaboration with researchers in other fields, such as education, psychology, or cognitive science. As someone with both technical and pedagogical expertise, I am uniquely qualified to contribute to these collaborations.

### 3.3 Software Engineering Research

There are several software engineering projects I'd like to explore:

- **Fault Tolerant Computing:** The “acceptability envelope” is a term used by Martin Rinard's research group at MIT to describe the tolerance each component of a system has for errors. By determining the tolerance for errors of each component of a system, developers can better decide where to focus their efforts as a deadline approaches. Using a bytecode transformer that ignores all null-pointer exceptions in a program, and then re-running all 150,000 snapshots of CS-2 code to see if any test cases pass that had previously failed, we can measure the acceptability envelope of the projects assigned to students, and eventually expand this work to include production codebases and their unit tests.
- **What is necessary to find bugs?** An important open question in the static bug finding community is, how much work do we need to do to locate bugs? Prior research has shown that FindBugs, using relatively simple intra-procedural analyses coupled with lightweight annotations for method parameters and return values, discovers a large number of bugs both in student code and in production-quality software. More powerful and

algorithmically more expensive techniques exist that can identify more complex bugs that Findbugs cannot. However, we lack a good understanding of the trade-offs between the fast, cheap approach of FindBugs and the slower, deeper approach used by projects Dawson Engler's Meta-Compilation system or the model-checker used by Compaq's Extended Static Checker (escJava) system. The Marmoset dataset of student code provides an opportunity to explore these trade-offs more fully before widening the domain to include production code.

## 4 Conclusion

I have the infrastructure to collect detailed data about *what* students do, and *how* they do it, when learning to program. Marmoset presents a unique opportunity for collaboration between Computer Science and the Learning Sciences to build a cognitive model of introductory programmers that helps answer *why* they do things in order to identify the most effective methods for teaching the material. While heavyweight institutions like Microsoft, Google and the NSF have recognized the importance of improving computer science education, this area has only recently received serious attention from academia. With the right team of researchers, there is a tremendous opportunity to break new ground and answer questions that are vitally important for training the workforce of the 21st century. I look forward to being a part of the team that tackles these issues.