Towards Improving Programming Habits to Create Better Computer Science Course Outcomes

Jaime Spacco
Computer Science Department
Knox College
Galesburg, IL 61401
jspacco@knox.edu

Davide Fossati
Computer Science Department
Carnegie Mellon University in Qatar
P.O. Box 24866
Doha, Qatar
dfossati@cmu.edu

John Stamper
Kelly Rivers
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
jstamper@cs.cmu.edu
krivers@cs.cmu.edu

ABSTRACT

We examine a large dataset collected by the Marmoset system in a CS2 course. The dataset gives us a richly detailed portrait of student behavior because it combines automatically collected program snapshots with unit tests that can evaluate the correctness of all snapshots. We find that students who start earlier tend to earn better scores, which is consistent with the findings of other researchers. We also detail the overall work habits exhibited by students. Finally, we evaluate how students use *release tokens*, a novel mechanism that provides feedback to students without giving away the code for the test cases used for grading, and gives students an incentive to start coding earlier. We find that students seem to use their tokens quite effectively to acquire feedback and improve their project score, though we do not find much evidence suggesting that students start coding particularly early.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Sciences Education—*Computer Science Education*

General Terms

Human Factors, Measurement

Keywords

computer science education, CS2, novice programmers

1. INTRODUCTION

In the United States, recent graduates with Computer Science degrees can expect high starting salaries [15] and excellent job prospects [14]. However, students earned about the same number of Computer Science Bachelors' in 2008 as in the mid 1980s [4], despite large increases in both overall college enrollment and the overall number of Bachelors' degrees awarded. The job market seems to be demanding students with the knowledge and skills taught in undergraduate Computer Science departments, yet students have not flooded into the CS major.

In fact, supplying the growing technological job market with knowledgable young computer scientists has proven quite difficult. Attrition rates in undergraduate CS programs are high [2], and there's evidence of a gap between what faculty think students should be able to do and what students actually can do [12]. High school students do not seem to know what computer science is [5], let alone show much interest in majoring in the discipline. Men earn about 80% of undergraduate computer science degrees in the United States [4], and this large gender gap is possibly keeping talented women out of a dynamic, exciting major.

Given that many introductory CS students struggle with learning to program, and that most attrition happens by the second year of study [2], helping students learn to program seems like a reasonable way to keep students in the major. There have been excellent recent efforts at collecting data to help us better understand novice compilation behavior[10, 17, 18], and novice programming habits [6, 13, 9]. This empirical work has allowed us to begin addressing key questions such as "How much time do students spend on programming assignments?", and "How much benefit do students get from starting early?". However, our understanding of novice programmers is still quite incomplete, and can be improved. We need a deep and nuanced understanding of how students learn to program before we can know where to make improvements.

This paper makes several contributions:

- We combine passively collected snapshots with unit tests to build a richly detailed portrait of student work habits.
- We find a statistically significant correlation between starting an assignment early and earning a higher score. This supports the conclusions of prior studies [8, 1, 7].
- We explore students' work habits in detail, examining what hours of day students work, how much work is done the day before and the day of the deadline, and the total amount of time spent coding on each project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITiCSE'13, July 1–3, 2013, Canterbury, England, UK.
Copyright 2013 ACM 978-1-4503-2078-8/13/07 ...$15.00.

course	CS-2
semester	Spring 2006
language	Java
# projects	6
# students	96
# snapshots	55,070
# compilable snapshots	37,401

Table 1: Overview of the Marmoset dataset.

- We examine students' use of release tokens, first introduced by the Marmoset data collection and automated testing system [16], showing evidence that suggests the tokens help students achieve higher scores without giving away the code to all of the test cases.

2. THE MARMOSET DATASET

In this work, we analyze data collected by the Marmoset system [16] in Spring 2006 during a CS2 course at the University of Maryland. Marmoset is an automated testing and data collection framework that combines three key components: a client-side data collection plugin for students' IDEs, a central server for testing student submissions against unit tests, and a novel token-based incentive system for revealing select feedback about the test cases (we give more details about *release tokens* in Section 2.2). The combination of fine-grained snapshots and detailed unit tests is particularly important, as we can run every snapshot against the unit tests for that project and pinpoint the *precise set of changes* that allowed a student to pass challenging test cases.

2.1 Snapshot Collection with the Eclipse Plugin

Marmoset uses an Eclipse plugin to capture snapshots of student files at each save event. Because Eclipse performs incremental compilation, the snapshots are not necessarily part of a traditional compile-edit cycle, producing a dataset qualitatively different than the compilation-based snapshot datasets collected by other researchers [10, 17, 13]. However, like these other datasets, our dataset is file-grained. Table 1 breaks down the overall statistics of the Marmoset snapshots.

2.2 Release Testing

Release tests were designed to achieve two goals: 1) Provide feedback about the correctness of student code *without giving away the code to the test cases*, and 2) Give students an incentive to start coding early. The first goal is to prevent students from simply "coding to the tests", while the second goal is a reaction to the widely held belief by faculty that students simply do not start coding early enough finish the project.

Students using Marmoset can upload their code to a central server, which executes their code against a series of unit tests. These test cases are partitioned into three categories:

- **Public tests**: The full source code of public tests is given to students with the project. Public tests are executed against all submissions to the server, and the results are always visible to students. All projects in this dataset had a couple of public tests to help students get started testing their code.
- **Release tests**: The source code of release tests is never revealed to students. Release tests are run against every submission, although students must spend a *release token* to receive feedback about the results of release testing. Students have three release tokens, and each token takes 24 hours to regenerate after use. In order to use a token, students must upload their code to the Marmoset server, authenticate themselves to the Marmoset server through a web interface, and then choose to spend a token. Using a release token reveals the number of release tests passed and failed, as well as additional feedback about the first two failed tests, such as the names of the test cases and the stack trace of any exceptions generated by the test.
- **Secret tests**: Neither the source code for the tests nor the outcome of running the test cases is revealed to students. Secret tests are equivalent to the de facto method of grading programming assignments for instructors who do not provide students with any pre-deadline feedback.

Theoretically, the scarcity of the release tokens should incentivize students to start working early, and to test their code thoroughly before spending a token. In our experiments, we examine students' use of release tokens over time.

2.3 Other Marmoset Details

The Marmoset snapshots used in this work are a rich, powerful resource in helping understand novice programmers. However, the data have a number of characteristics, as well as key limitations, which should be kept in mind when assessing this work:

- There have not been any controlled studies of the release token mechanism, so it is very difficult to draw any strong conclusions about the effectiveness of this system. The lack of controlled studies is due to logistical and policy hurdles rather than any fundamental limitation of the system.
- The automated grading makes extensive use of unit tests, which means that students are programming to an API rather than writing code from scratch. We cannot use this data to evaluate students' design abilities, or to understand how they would program when beginning with a blank screen.
- The dataset is from a CS2 course, so the students are not true novices. Attrition rates are typically lower in CS2 than in CS1, so this data may serve more use in determining what helps students perform better. It is worth noting that Marmoset has been used before in CS1 courses, though we do not have access to the data collected.
- The dataset only includes scores for the functional correctness of a project as computed by the unit tests. Each assignment was also assessed manually for style, but we do not have access to those scores, nor do we have the students' exam scores. Thus, we are only able to consider the students' ability to write functionally correct code.
- The data collected only captures the amount of time students *actively spend programming*. We are unable

to measure the time spent reading the textbook, outlining the program on paper or the whiteboard, reading online resources, or any of the other work that contribute to a successful programming assignment. This limitation is not unique to Marmoset; all automatic snapshot capture systems are limited to data they can easily collect.

3. RELATED WORK

Some work has already been done in analyzing large corpuses of student data in order to find useful trends and correlations. For example, several researchers have examined compilation behavior, in order to determine how students react to errors and which errors are most common [10, 9]. Correlations have also been found between students' relationships with errors and their midterm scores which can be used to identify at-risk students [17]. Some researchers look more at how student behaviors change over time, using Bayesian learning models [11] and learning analytics through visualizations [3].

Closer to our area of study, analysis has also been done on student submission behavior and how it affects learning. Some researchers have conducted work very similar to ours, using submission data from Web-CAT to find that students who submitted more and earlier were less likely to fail the assignment [1, 7]. In addition, it has also been shown that students who submit late on the first assignment are more likely to continue submitting late and do more poorly in the class [8]. Fenwick et al have also found trends suggesting that students who program in more 'sessions' (doing more incremental work) are more likely to achieve higher grades [9]. We aim to replicate and expand on this work, while also more closely examining the effects of hidden tests and release tokens.

4. EXPERIMENTS

The Marmoset dataset combines snapshots with unit tests to give researchers a detailed view into student work patterns (i.e. when they are working, what they are changing) and the correctness of their code (i.e. what test cases are they passing). Such fine-grained data allows us to see the precise snapshot where students have a breakthrough in the correctness of their program, as well as what they were doing before and what they do after. By grouping the snapshots into estimated "work sessions", we can look at factors within a work session that increase or decrease the score achieved at the end of the session.

4.1 Overview of the Data

The dataset has over 37,000 compilable snapshots from 96 students over six CS2 projects. Table 2 shows the mean and median scores for each project, as well as the number of students with at least one submission for each project (submitting to the server is required for credit on a project; the snapshots are purely for research and automated backups). Students receive zero points if they fail to make a submission for a project; the true mean and median values would be slightly lower if we included those default scores of zero.

Figure 1 shows the distribution of snapshots over the hours of the day. We see a lot of activity between 4pm and 6pm (16:00 to 18:00). We initially thought this was because deadlines were always at 6pm and we were seeing a rush to complete the project; however, we looked at only the snapshots that occurred more than 24 hours before the deadline, and found essentially the same shape as in Figure 1. Thus, if students prefer working 4pm to 6pm, then setting the deadline a couple of hours later might allow students to work at their preferred time without the added pressure of an impending deadline.

Project	median	mean	# students w a submission
1	100%	90.8%	90
2	82.9%	60.2%	89
3	95.0%	82.2%	90
4	92.2%	82.2%	85
5	100%	85.3%	84
6	90.0%	80.1%	81

Table 2: For each project, we report the mean and median scores, as well as the number of students (out of 96) who made at least one submission.

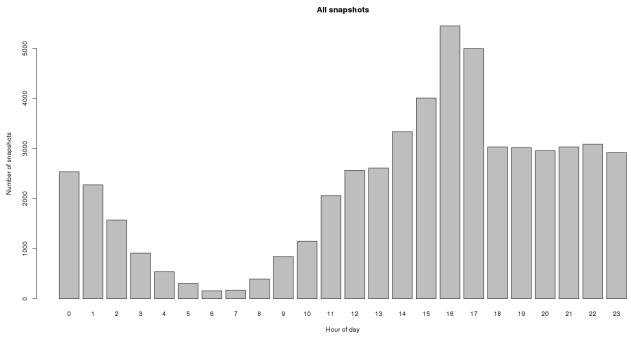

Figure 1: Number of snapshots distributed over hours of the day. Note the spike in activity between 4pm and 6pm.

Figure 2 shows the number of snapshots per day in the 10 days leading up to each project deadline (students typically have 10 days to complete a project). The bar for each day is broken up into *positive* and *neutral or negative* snapshots. Positive snapshots increase the number of passing test cases, and hence the score, from the previous snapshot, while neutral/negative snapshots either make no improvements, or decrease the score. We see a lot of improvements made the day before, and the day of, the project deadline, which suggests that many students are doing substantial work in the 48 hours before the deadline. This may mean that students are not, in fact, starting work early enough. We more work and more improvements 5 days before the deadline that we do 3 or 4 days before. We believe this occurs because four out of six deadlines were on a Thursday, meaning students worked more on Saturdays than on Sundays.

Figure 3 compares each student's start time (i.e. the time before the deadline of their first snapshot, to the nearest hour) to the final score they ultimately achieved. The relationship is statistically significant, and gives a clear indication that starting an assignment correlates with better scores. Note that the test cases may not be exhaustive, and that there may be differences in quality or correctness be-

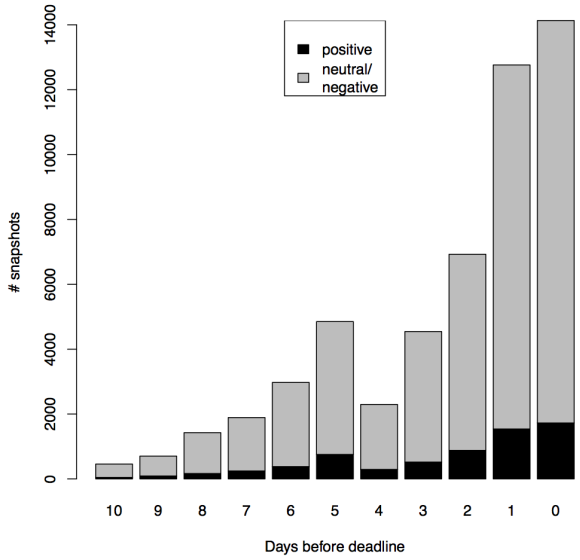

Figure 2: Number of snapshots per day leading up to the deadline. Positive snapshots are snapshots that increase the score from the previous snapshot (i.e. a positive delta), while negative or neutral snapshots have an identical or lower score than the previous snapshot.

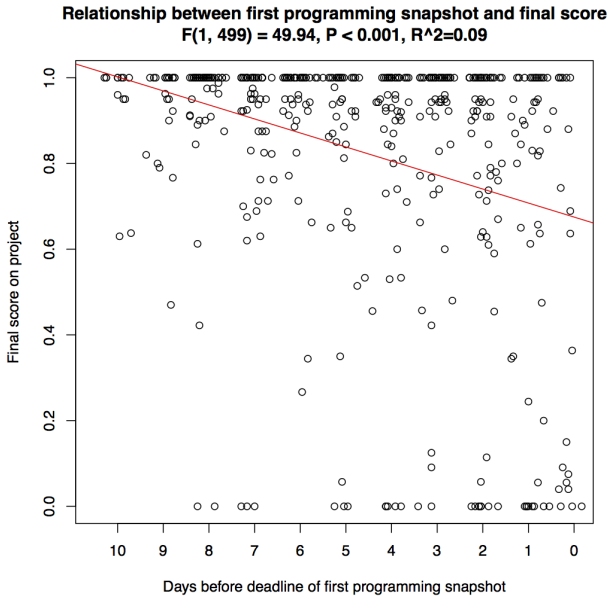

Figure 3: The X-axis shows the time of the first snapshot (i.e. when the student began writing the program), while the Y-axis shows the final score the student eventually achieved.

tween the programs that earned 100% that these test cases are unable to identify.

4.2 Work Sessions

Marmoset collects a snapshot whenever students save their files, so snapshots are not independent events. Two students may save files at very different intervals due to personal idiosyncrasies that have nothing to do with their programming ability. Thus we have to take care when aggregating information about snapshots.

In order to get a more accurate picture of student work habits, we have clustered each student's snapshots into estimated "work sessions" using the very rough heuristic that snapshots less than 20 minutes apart are part of the same work session. We have not observed the work habits of any of the students in this data set, so we lack a good understanding of student work habits needed to validate our heuristic. However, we expect that even an approximate clustering of the snapshots will give us a more accurate picture of student work patterns than treating each snapshot independently would. Figure 4 shows the relationship between the estimated time spent programming in each session and the *score differential* (i.e. the difference in the score between the first and last snapshot of a work session). There is a statistically significant relationship, though the correlation is fairly weak.

Figure 5 shows the relationship between the total estimated time spent programming on each project and the final score on the project. The relationship, while statistically significant, is relatively weak, and could simply indicate that students who spent only a few minutes on an assignment end up with extremely low scores.

4.3 Release Tokens

Students using Marmoset are given a set of public unit tests, while other unit tests, called *release tests*, are stored on the server. Students can upload their code to the server, spend a *release token*, and see the number of release tests passed and failed, as well as additional feedback about the first two release tests which failed. Furthermore, students only have 3 release tokens, and each token takes 24 hours to regenerate.

The goals of release tests are to give helpful feedback without giving away the test cases, and to create an incentive for students to start programming assignments early. We find, however, that students only used an average of 2.4 release tokens per assignment, and that about 55% of the total release tokens used were deployed on the day of the deadline, or the preceding day. These data suggest that students are using release tests to get feedback, but are not necessarily starting early to use more than three release tokens.

Figure 6 shows the relationship between the number of release tokens used per work session (maximum of 3) and the difference between the score at the beginning and end of the session. The relationship is statistically significant, though the model does not explain much of the variance.

Table 3 splits up the work sessions by whether they use at least one release token, then by whether they increase (positive), decrease (negative), or have no effect (neutral) on the score. In other words, a positive work session means that the score for the snapshot at the end of the session is higher than the score for the snapshot at the beginning. The differences are statistically significant ($p < 0.001$) according to a Chi Squared analysis.

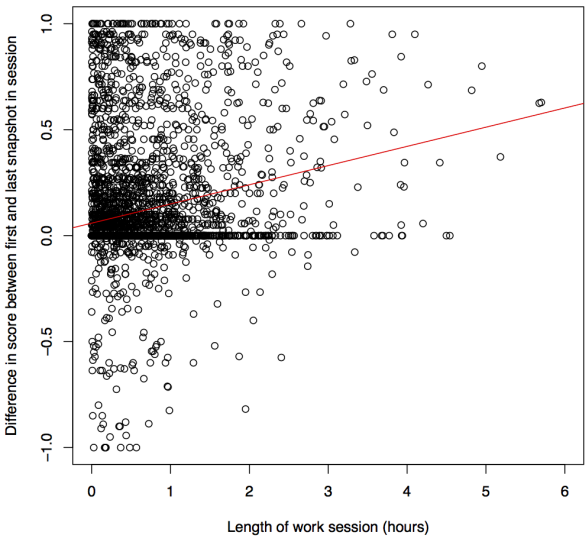

Figure 4: Relationship between the length of estimated work sessions and the change in project score from the start to the end of the session.

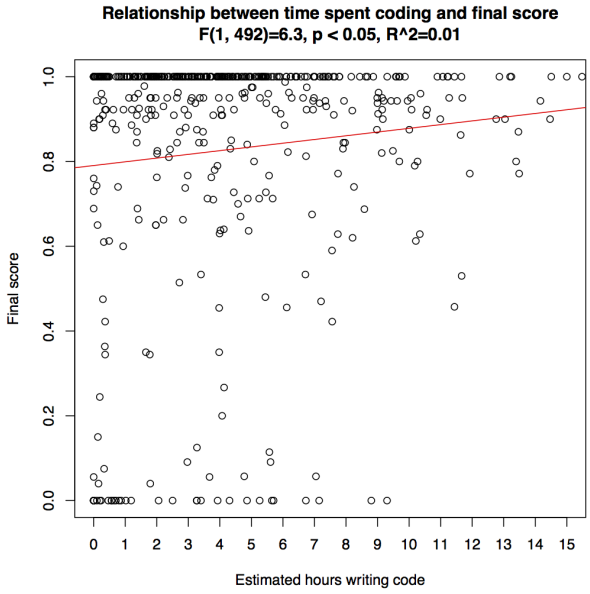

Figure 5: Relationship between the total estimated time spent writing code and the final score on the project.

	positive	neutral	negative	total
used >= 1 token	515	470	36	1021
no tokens used	959	2129	175	3263
total	1474	2599	211	4284

Table 3: Work sessions divided up, first by whether they increase (positive), decrease (negative), or have no effect (neutral) on the score, then by whether at least one release token was used in the session.

5. CONCLUSIONS AND FUTURE WORK

Combining automatically collected program snapshots with unit tests has given us a detailed picture of student work patterns, revealing some obvious facts (i.e. students do very little work between 4am and 8am), as well as some less obvious facts (i.e. students work more on Saturdays than on Sundays and more between 4pm and 6pm than any other 2 hour block of time).

Unsurprisingly, we find that students do the majority of their work within 48 hours of the deadline, and that students who start earlier tend to earn better grades, which supports the findings of previous researchers [8, 1, 7]. We found a weak relationship between the total amount of time coding and the final score, though this may simply reflect the fact that even the best students must spend a minimum amount of time coding the solution.

We also examined the use of release tests in more detail. We found that students used an average of 2.4 release tests per project, that work sessions that used one or more release test were statistically more likely to improve the student's score than work sessions that do not use a release token, and that using additional release tokens in a work session tends to improve the session. This evidence suggests that students are effectively using their release tokens to obtain feedback and improve their scores. In turn, this suggests that the release tokens are indeed providing valuable feedback to students *without revealing the code to the test cases*. This achieves the first design goals of release tests: To provide limited but useful feedback to students. Without a controlled study of release tokens, it will be difficult to show that the release tokens are achieving the second design goal: influencing students to start coding earlier. The data in paper, namely that students use an average of 2.4 tokens per project and that 55% of release tokens are used in the last two days before a deadline, suggest that students are *not* starting their projects particularly early. In fact, it's possible that the tokens have the unanticipated side-effect of *discouraging students from writing their own test cases*, since even procrastinators who start coding the same day that the project is due know that they have at 3 tokens to use. We know that the instructors who regularly use the system eventually *decreased* the number of release tokens available from 3 to 2 in order to further discourage procrastination.

In the future, we hope to conduct a controlled study of release tokens, where an experimental group has release tests and release tokens, while the control group has secret tests. This study would be a tremendous help in understanding how students respond to the incentives offered by release tokens.

6. ACKNOWLEDGMENTS

The authors would like to thank Bill Pugh and Nelson Padua-Perez for collecting the data used for this study in their course, and the Pittsburgh Science and Learning Center's LearnLab Summer School for bringing the authors together.

7. REFERENCES

[1] A. Allevato, M. Thornton, S. H. Edwards, and M. A. PÃl'rez-QuiÃśones. Mining data from an automated

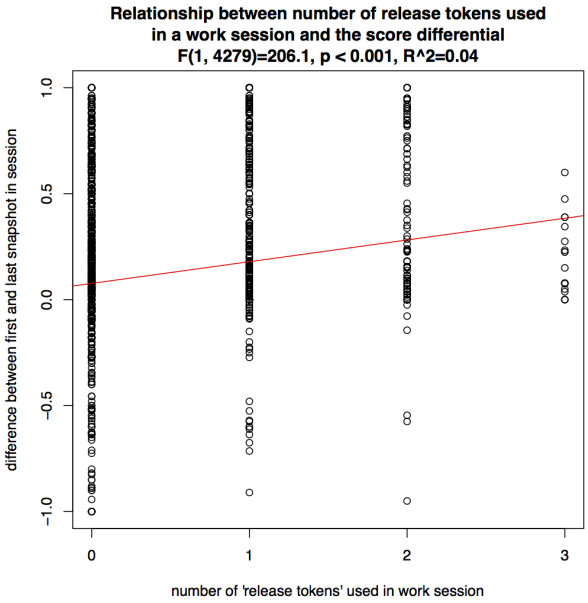

Figure 6: Relationship between release tokens used in a work session and the score differential of the work session.

grading and testing system by adding rich reporting capabilities. In R. S. J. de Baker, T. Barnes, and J. E. Beck, editors, *EDM*, pages 167–176. www.educationaldatamining.org, 2008.

[2] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bull.*, 37(2):103–106, June 2005.

[3] P. Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, LAK '11, pages 110–116, New York, NY, USA, 2011. ACM.

[4] N. S. Board. Science and engineering indicators digest 2012. 2012.

[5] L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. *SIGCSE Bull.*, 38(1):27–31, Mar. 2006.

[6] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. *SIGCSE Bull.*, 36(1):26–30, Mar. 2004.

[7] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the fifth international workshop on Computing education research workshop*, ICER '09, pages 3–14, New York, NY, USA, 2009. ACM.

[8] N. Falkner and K. Falkner. A fast measure for identifying at-risk students in computer science. In *Proceedings of the ninth annual international conference on International computing education research*, pages 55–62. ACM, 2012.

[9] J. B. Fenwick, Jr., C. Norris, F. E. Barry, J. Rountree, C. J. Spicer, and S. D. Cheek. Another look at the behaviors of novice programmers. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 296–300, New York, NY, USA, 2009. ACM.

[10] M. C. Jadud. A first look at novice compilation behaviour using bluej. *Computer Science Education*, 15:1–25, 2005.

[11] J. Kasurinen and U. Nikula. Estimating programming knowledge with bayesian knowledge tracing. In *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '09, pages 313–317, New York, NY, USA, 2009. ACM.

[12] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *SIGCSE Bull.*, 33(4):125–180, Dec. 2001.

[13] C. Murphy, G. Kaiser, K. Loveland, and S. Hasan. Retina: helping students and instructors based on observed programming activities. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 178–182, New York, NY, USA, 2009. ACM.

[14] P. C. of Advisors on Science and Technology. Designing a digital future: Federally funded research and development in networking and information technology. 2013.

[15] N. A. of Colleges and Employers. Nace january 2013 salary survey. 2013.

[16] J. Spacco, J. Strecker, D. Hovemeyer, and W. Pugh. Software repository mining with marmoset: an automated programming project snapshot and testing system. In *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.

[17] E. S. Tabanao, M. M. T. Rodrigo, and M. C. Jadud. Predicting at-risk novice java programmers through the analysis of online protocols. In *Proceedings of the seventh international workshop on Computing education research*, ICER '11, pages 85–92, New York, NY, USA, 2011. ACM.

[18] I. Utting, N. Brown, M. Kölling, D. McCall, and P. Stevens. Web-scale data gathering with bluej. In *Proceedings of the ninth annual international conference on International computing education research*, ICER '12, pages 1–4, New York, NY, USA, 2012. ACM.