Analyzing Student Work Patterns Using Programming Exercise Data

Jaime Spacco
Knox College
jspacco@knox.edu

Paul Denny
University of Auckland
paul@cs.auckland.ac.nz

Brad Richards
University of Puget Sound
brichards@pugetsound.edu

David Babcock
David Hovemeyer
James Moscola
York College of Pennsylvania
{dbabcock,dhovemey, jmoscola}@ycp.edu

Robert Duvall
Duke University
rcd@cs.duke.edu

ABSTRACT

Web-based programming exercises are a useful way for students to practice and master essential concepts and techniques presented in introductory programming courses. Although these systems are used fairly widely, we have a limited understanding of how students use these systems, and what can be learned from the data collected by these systems.

In this paper, we perform a preliminary exploratory analysis of data collected by the CloudCoder programming exercise system from five introductory courses taught in two programming languages across three colleges and universities. We explore a number of interesting correlations in the data that confirm existing hypotheses. Finally, and perhaps most importantly, we demonstrate the effectiveness and future potential of systems like CloudCoder to help us study novice programmers.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Measurement

Keywords

programming exercises; student work patterns; outcomes

1. INTRODUCTION

Many students struggle in introductory computer science courses [4]. Web-based programming exercise systems, such as CodingBat [16] and CloudCoder [10], can be a useful way for instructors to provide students with additional opportunities to practice basic concepts and techniques with the ultimate goal being (hopefully) to help students achieve better outcomes.

In these systems, an exercise requires the student to write a small amount of code (perhaps 5–15 lines) to perform a specified computation. The correctness of the student's code is judged automatically by running the code against a set of tests. If all of the tests pass, the student's code is judged to be correct. Because these systems are web-based and the assessments are automatic, students can work wherever and whenever is convenient for them, and receive immediate feedback. This combination of features allows students to receive more feedback for a greater number of homework exercises than would be possible with manually graded assignments.

In addition to the pedagogical benefits of using programming exercises, the data collected by programming exercise systems offers a detailed window into the students' habits. In this paper, we examine data collected using the CloudCoder programming exercise system in five introductory programming courses taught at three different institutions.

This paper makes the following contributions:

- We demonstrate the research potential of collecting and analyzing data from short programming exercise systems such as CodingBat and CloudCoder.
- We find a positive and consistent, although weak, correlation between a student's effort and success on CloudCoder exercises and the student's final exam score.
- We find that more difficult exercises (as measured by the average score achieved by students) require more time and effort to complete, but are not more likely to result in compilation failures.
- We explore possible indications of student struggling (*flailing*), and find that the amount of time spent on an exercise, as well as the frequency of submissions, are weakly inverse-correlated with the chance the code compiles. However, time spent and the frequency of

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.
Copyright © 2015 ACM 978-1-4503-2966-8/15/03 ...$15.00.
http://dx.doi.org/10.1145/2676723.2677297 .

submissions are *not* consistently correlated with overall success on an exercise.

- We find, unsurprisingly, that over the course of the term, the assigned exercises become more difficult, and the students become more adept at writing syntactically correct programs.

2. RELATED WORK

Numerous hurdles confront students learning to program. Many novice programmers struggle to interpret cryptic compiler error messages and fail to overcome syntax errors [12]. Logic errors generally present an even greater challenge for novices [9], and it is known that poor debugging skills can lead to the introduction of new errors, frustration, and ultimately failure in a course [3, 13].

A wide variety of tools have been developed by computing education researchers to help novice programmers. These tools can also assist researchers, as they can be used to collect data on student behavior, shedding light on the kinds of problems that students encounter and revealing strategies that prove effective. Collecting such data over an extended period of time for research purposes is one of the explicitly stated goals of the Blackbox project outlined by Brown et. al. Blackbox collects information about submissions made by students in the BlueJ environment into a central repository [5]. Given the wide adoption of BlueJ in introductory programming courses around the world, Blackbox data could potentially form the basis for vast cross-institutional studies.

Even at more modest scales, the BlueJ environment has been a rich source of data on novice programmer behavior. Jadud monitored student compilation events within individual BlueJ work sessions with a particular focus on syntax errors [11]. Certain kinds of errors occurred frequently, directing future work efforts toward helping students break out of repetitive error cycles. More fine-grained data, including basic project and file management within BlueJ, was collected by Norris et. al. using their ClockIt data logger [14]. Analysis of this data revealed that students who started work on assignment tasks earlier tended to obtain superior results, echoing the advice of many an instructor.

Automated grading tools have been another source of data used by researchers to explore student work habits. Edwards et. al. analyzed data from Web-CAT, an online tool where students iteratively submit both their source code and the set of tests used to verify their work [8]. Although student behavior was inconsistent over the course of a semester, they found that in general, better outcomes were achieved when students started working on assignments early. A similar result, showing that students who started work earlier tended to earn higher scores, was reported by Spacco et. al. from an analysis of student data collected by the Marmoset automated grading system [17].

Previous work has also examined novice behavior when solving short programming exercises, similar to those found in CodingBat [16] and CloudCoder. An example is the work of Denny et. al. with the CodeWrite tool where students create, as well as solve, the exercises [7]. Their analysis revealed that certain syntax errors consume a great deal of student time and cause significant trouble, prompting a new tool design displaying simplified error messages and associated examples [6]. In this paper, to better understand student work habits when solving short programming exercises, we explore activity data collected by the CloudCoder tool.

3. CloudCoder

CloudCoder [1, 15] is an open source, web-based programming exercise system inspired by CodingBat. It supports exercises in several programming languages, including (at the time of writing) C/C++, Java, Python, and Ruby. The system utilizes a repository of freely-redistributable exercises [2] contributed by users.

One of the goals of the CloudCoder project is to provide a platform for collecting data on how students learn to program. In support of this goal, CloudCoder collects a rich stream of data related to the process students undertake when working on programming exercises. The specific data we analyze in this paper are as follows.

Submissions When working on an exercise, students can click the Submit button to have their code compiled and tested on the server. For each submission event, CloudCoder records the full text of the submission and the result of compiling the submission. Students may submit as many times as they wish without penalty with the system using the best submission as their final score. We compute the best submission as simply the percentage of test cases the student code passes (i.e., 8 test cases passed out of 10 total test cases is 80%).

Test results If a submission compiles successfully, it is executed against each of the exercise's tests and records the test results. Possible results from executing a test are **passed** if the code's result matches the expected result, **failed** if the code's result does not match the expected result, or **exception** if the code misbehaves such as encountering a fatal runtime error or exceeding allowed CPU time.

Code edits As students edit code within CloudCoder, the underlying code editor produces a stream of insertion and deletion events which are captured and recorded. The events are fairly fine-grained: for example, as students type, the insertion events record individual characters.

All of the data described above are recorded with millisecond resolution timestamps, allowing detailed time-based analyses to be performed.

4. ANALYZED DATA SETS

We analyzed data sets collected from five courses taught at three institutions in 2013–2014. An overview of the data used in our research is available in Table 1.

4.1 CS 101 at York College

The two CS 101 course offerings at York College were CS 1 courses taught using C. CloudCoder exercises were made available to accompany the reading assignments, for exam review, and as in-class quizzes and labs. In both semesters, most of the exercises were optional, although the instructors strongly encouraged students to do them. The extent to which different students attempted to complete the exercises varied considerably.

The format of the courses differed somewhat in the two semesters. In both semesters, a substantial chunk of each class meeting was spent on in-class lab activities. In Spring 2013 each class began with a traditional lecture. In Spring

School	Term	Course	Language	# stud.	total # exercises	avg. # started	avg. # finished
York	Spring 2013	CS 1	C	133	62	37	33.9
York	Spring 2014	CS 1	C	86	53	30.2	27.4
Auckland	Fall 2013	Programming for engineers	C	740	12	11.7	11.6
Duke	Fall 2013	CS 1	Python	233	62	44.2	38.2
Duke	Spring 2014	CS 1	Python	194	55	45.2	42.2

Table 1: Overall breakdown of CloudCoder data for the three institutions and five courses.

		Auckland	York 2013	York 2014	Duke 2013	Duke 2014
num. completed vs. final exam	p-value	< 0.001*	0.083	0.004*	< 0.001*	< 0.001*
	R^2	0.089	0.030	0.119	0.231	0.386
num. attempted vs. final exam	p-value	< 0.001*	0.370	0.006*	< 0.001*	< 0.001*
	R^2	0.073	0.008	0.106	0.194	0.382
pct. completed vs. final exam	p-value	< 0.001*	< 0.001*	0.041*	< 0.001*	< 0.001*
	R^2	0.082	0.138	0.06	0.149	0.295

Table 2: Results from linear regressions. All regressions try to predict the final exam score, and all significant results show the score on the final to increase with additional work on CloudCoder. Statistically significant results are marked with a star (*).

2014 we used a fully flipped-classroom approach, with each class meeting starting with a peer instruction quiz. The students in CS 101 consisted of Computer Science, Electrical and Computer Engineering, and (in Spring 2013) Mechanical Engineering majors, along with a small number of Mathematics majors. There were also a small number of students from other majors. The decrease in class size from Spring 2013 to 2014 is due to the Mechanical Engineering program starting its own programming course in Spring 2014.

The "final" exam scores reported for both semesters of data from York are not the official final exam, but rather the last of four midterm exams. The last midterm exam took place during the last week of classes, contained only programming questions where students were asked to write code, and was cumulative; however, some of the questions emphasized topics from later in the course, such as struct types, and so may not necessarily be representative of a traditional final exam. The final exam at York is optional for students who have already achieved their desired grade, and so we have chosen not to include it.

4.2 ENGGEN 131 at University of Auckland

Over 700 students took Engineering 131 at the University of Auckland in the Fall 2013 semester. The course, required of engineering majors, taught two programming languages, C and Matlab. CloudCoder was used for one assignment, containing 12 exercises and worth about 2% of the marks for the course, during the portion of the course taught in C. CloudCoder does not support Matlab, so there were no CloudCoder exercises for that portion of the course. The final exam contained both multiple-choice and code-writing questions covering both C and Matlab. Unfortunately, we do not have the breakdown of which exam questions covered which language.

4.3 CompSci 101 at Duke University

Computer Science 101, Duke's CS 1 course, is taught using Python. Both semesters of the course included in our dataset regularly assigned CloudCoder exercises, worth about 10% of the semester grade. The final exams were traditional written exams containing about half multiple-choice and half code-writing questions; however, we only have the aggregate scores.

5. DATA ANALYSIS

The obvious question is whether CloudCoder exercises help students learn to program. Thus far, we have not been able to perform a large-scale controlled study to address this question directly. Instead, we have performed a post-hoc, exploratory analysis of the data collected by CloudCoder.

5.1 CloudCoder Exercises and Exams

To explore the relationship between CloudCoder exercises and final exam scores, we performed linear regressions of the number of CloudCoder exercises attempted and completed, as well as the percentage of exercises attempted that were completed, against the final exam score for each course in our dataset. We also compare the percent of submissions that compile for each user against their final exam score. Table 2 shows the p-value, and the coefficients of determination (R^2) for these linear regressions.

CloudCoder exercises show the strongest predictive power for the final exam at Duke, and the weakest predictive power at Auckland. This makes sense, as University of Auckland students only completed one 12-exercise CloudCoder assignment, and the final exam covered Matlab (for which there were no online exercises) as well as C; students at Duke University, on the other hand, completed regular CloudCoder assignments, and the final exam asked programming questions that were similar to these exercises.

The data from York College is mixed; it is unclear why the number of exercises attempted and answered was statistically correlated with the exam scores in 2014 but not 2013. One possible explanation is that the CloudCoder exercises were not as well integrated into the course in 2013, given that it was the first full-scale effort to include them. Another possibility was that the student population changed signif-

		Auckland	York 2013	York 2014	Duke 2013	Duke 2014
avg. num. sessions vs. avg. best score	p-value	< 0.001*	< 0.001*	< 0.001*	< 0.001*	< 0.001*
	R^2	0.702	0.770	0.593	0.218	0.654
pct. compilable subs. vs avg. best score	p-value	0.525	0.120	0.109	0.197	0.180
	R^2	0.042	0.040	0.050	0.028	0.034

Table 3: Results from linear regressions against average best score achieved by all users who attempted each exercise. Statistically significant results are marked with a star (*). Work sessions are computed by clustering keystrokes less than 10 minutes apart into work sessions.

		Auckland	York 2013	York 2014	Duke 2013	Duke 2014
total time in mins. vs. pct. compilable subs.	p-value	< 0.001*	< 0.001*	< 0.001*	< 0.001*	< 0.001*
	R^2	0.090	0.007	0.008	0.009	0.019
avg. # subs./min. vs. % compilable subs.	p-value	< 0.001*	< 0.001*	< 0.001*	< 0.001*	< 0.001*
	R^2	0.195	0.026	0.020	0.024	0.020
total time in mins. vs. best score	p-value	0.002*	< 0.001*	< 0.001*	0.034*	0.417
	R^2	0.001	0.004	0.015	< 0.001	< 0.001
avg. # subs./min. vs. best score	p-value	0.507	0.072	0.405	0.017*	0.435
	R^2	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001

Table 4: Results from linear regressions of the total time spent on each exercise, and the average number of submissions per minute for each exercise, against the percent of submissions that compile, and the best score achieved on each exercise. Each data point represents all combined work sessions for a student on a particular problem. Statistically significant results are marked with a star (*). Work sessions are computed by clustering keystrokes less than 10 minutes apart into work sessions. The best score is the maximum score achieved by the student on any submission.

icantly in 2014, with Mechanical Engineering students departing. Finally, there was a significant pedagogical change in 2014, with the course being fully "flipped". In any case, the discrepancy highlights the difficulty of post-hoc analysis.

With the exception of the Spring 2013 York data, all of the relationships indicated in Table 2 are significant, although relatively weak (R^2 at most 0.38). Overall, this suggests a positive relationship between a student's engagement with CloudCoder exercises and their final exam score.

However; we must urge caution when interpreting these results. We do not know what would have happened had these courses been taught *without* CloudCoder, and so we cannot conclude that it was CloudCoder that helped, as opposed to doing homework in general. We hypothesize that CloudCoder's automation enables instructors to assign more practice exercises, which leads to more students learning, but we do not have the data to support this claim.

5.2 Evaluating CloudCoder Exercises

Table 3 explores, over each exercise, predictors of the average best score achieved by each user who attempted the exercise. We compute *average best score* by summing each student's best score on the exercise, and then dividing by the number of students. As might be expected, more difficult exercises (as evidenced by lower average best scores) strongly correlate with more effort by students, as measured by additional work sessions: i.e., students were unable to complete the exercise in a single sitting. We report regression results for the average number of work sessions for each exercise; however, we should note that many other ways of measuring effort, such as average number of minutes per user per problem, average number of keystrokes, etc., also show similar results.

We also note that there is no statistically significant relationship for any of our datasets between the percentage of submissions that compile and the average best score achieved. In other words, for the more difficult exercises, it is not more difficult for students to submit syntactically valid code.

5.3 Students Struggling in CloudCoder

One intriguing use of systems such as CloudCoder would be to detect "flailing" [9] students early enough in the term that the instructor might intervene. However, this begs the obvious question: What does *flailing* look like (i.e., students who are working unproductively, and are unlikely to make progress), and what does *learning* look like (i.e., students who are working productively)? Two common assumptions that all of the authors of this paper have heard from colleagues are 1) sometimes a student is unlikely to complete an exercise, even if given an infinite amount of time, and 2) small changes made in rapid succession with little thought are unlikely to represent productive work. We chose two metrics to represent these concepts: total amount of time spent on an exercise, and average number of submissions per minute of work.

Note that both of these metrics rely on total work time, which we compute by clustering all keystrokes within 10 minutes into the same work session. We acknowledge that this sort of clustering is course-grained, and that the 10-minute cut-off is subjective. Determining an accurate cut-off time by observing students as they work remains future work.

We present our analysis of student struggles in Table 4. Both the total amount of time spent on an exercise, and the number of submissions per minute of work, are inversely correlated with the chances that a submission would compile. In other words, the more time a student works on an exercise, or the more times they submit per minute, the less likely those submissions are to compile. However, other than at Auckland, the R^2 values are quite low, which implies that many other factors contribute to the compilability of a student's submissions.

Our potential measurements of flailing are less effective at predicting a student's best score. Three of five datasets show a statistically significant inverse relationship between minutes of work and the best score, but the relationship is weak; Duke 2013 actually shows a significant, extremely weak, *positive* relationship between minutes and best score, while Duke 2014 shows no statistical relationship. Given these results, it is hard to infer a relationship between time spent working on an exercise, and the best score achieved on that exercise. We also see essentially no relationship between the number of submissions per minute, and the best score achieved, as our only significant result is extremely weak.

One possible interpretation of the data is that short, quick code changes mean something different when the code compiles than when it does not. This data could occur because the immediate feedback from CloudCoder allows students to diagnose and fix problems quickly; however, the data could also happen because a guess-and-check style of coding is, in fact, an effective strategy for short exercise systems. Similarly, additional minutes of work may mean something different when the code compiles than when it does not, presumably because students are working on *semantics* rather than *syntax*.

These interpretations are exploratory and highly speculative, and we must stress that we cannot draw any strong conclusions without more data and a much more nuanced analysis of the data.

5.4 Improvement Over Time

Another question is whether students improve during the semester. To shed light on this question, for each student, we number the exercises in the order in which each student attempted them. The ordering is relative to each student, which means that exercise #1 may not be the same exercise for all students, because students may choose to do the exercises for an assignment in any order, may skip exercises, and may work on multiple exercises at the same time. However, even given these complications, the results in Figure 2 show a striking pattern. We can see very clearly that the exercises assigned at York College and Duke University become progressively more difficult, as evidenced by the lower average best scores achieved on later exercises (the Auckland data was a single assignment, so it is not surprising that the order in which students did the exercises mattered less than at other schools). At the same time, we clearly see that, other than the York 2013 data, the more exercises a student attempts, the *higher* the chance that future exercises will compile, which seems to show clear improvement in compilations. We are unsure why the data from York 2013 suggests that *more* practice somehow makes students worse at compiling their code. One possibility is that stronger students became less likely to do the exercises over time.

Interestingly, the data from Figure 2 suggests that students improve at compiling with practice, and that the improvement happens rapidly for C as well as Python. However, Figure 1 still reveals differences in the overall likelihood that a submission compiles between the datasets, with code written in C less likely to compile than code written in

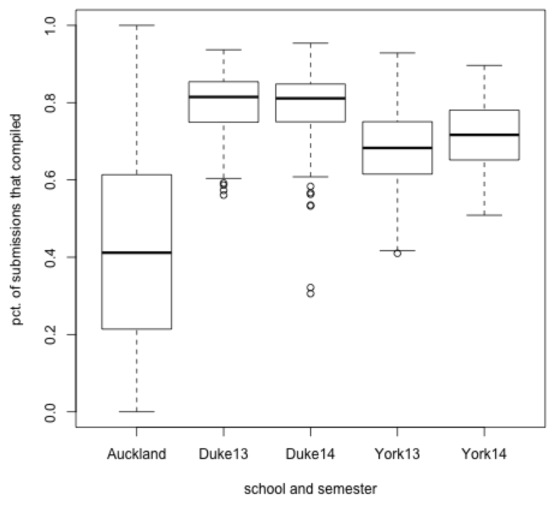

Figure 1: Boxplot of compilation percent for the five datasets. The box depicts and first and third quartiles, while the whiskers are 1.5 × IQR (interquartile range). Outliers are plotted as points.

Python. Given that many schools have switched to Python for the first programming language, at least in part due to its "easier syntax", an open question is whether the choice of language shows a measurable effect on the rate at which students learn to fix syntax errors. This is an extremely difficult question to address, as many things vary between courses taught at different institutions. A very careful and detailed analysis would be required even to begin to answer this question.

6. DISCUSSION AND FUTURE WORK

In this work, we have explored a number of interesting correlations and relationships in data collected from hundreds of novices learning to program in two different programming languages, at three different institutions.

This work represents a first look at the rich dataset we have amassed, and only scratches the surface of possible research angles. We hope to address a number of interesting questions in future work. For example, our current work only includes students who eventually took the final exam. Are there measurable differences in behaviors or outcomes, especially in the first three or four weeks of the term, between students drop the course and those who do not? Coding style is another interesting angle of study: Does code that better follows stylistic guidelines or conventions, perhaps as evaluated with a static style checker, correlate with better outcomes, and does style matter as much for shorter CloudCoder exercises as it does for longer, "nifty" assignments? Finally, there is still much to be learned about compilation and syntax errors for novices.

7. ACKNOWLEDGMENTS

This work was supported by a SIGCSE Special Projects Grant. Cloud-hosting was supported by an Amazon Web

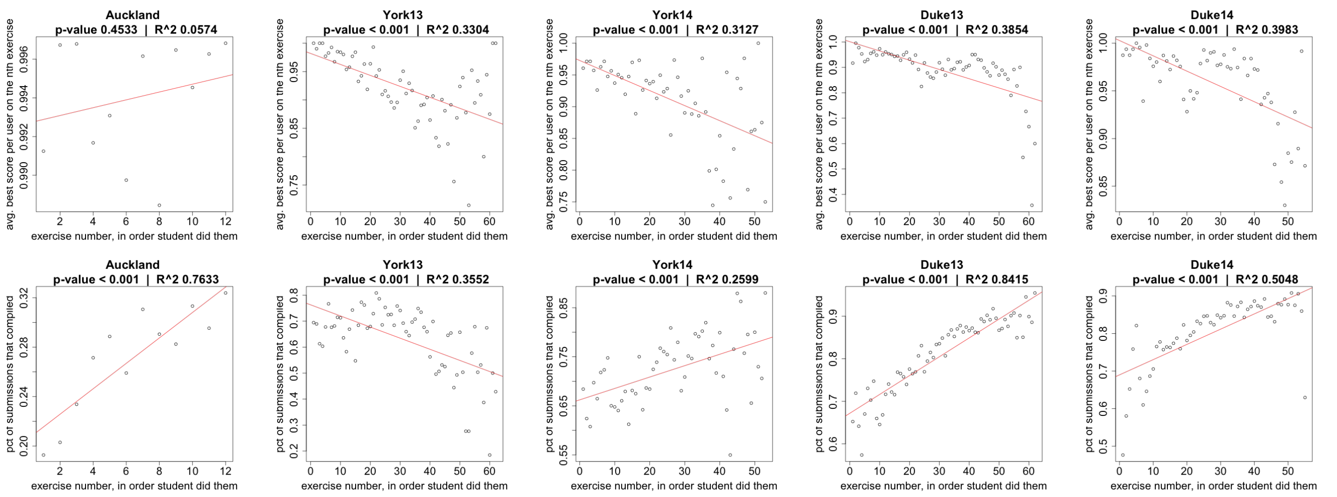

Figure 2: Results from linear regressions. The x-axis is the exercise number from the perspective of each student, i.e., 1 is the first exercise the student started working on, 2 is the second exercise, and so on. Regressions try to predict the average of the best scores achieved by each student, or the percentage of compilable submissions, for each exercise.

Services (AWS) in Education Grant. Thanks to Kevin Hastings for help with statistics.

8. REFERENCES

[1] Cloudcoder - welcome to cloudcoder! http://cloudcoder.org, June 2014.

[2] Cloudcoder exercise repository. https://cloudcoder.org/repo, June 2014.

[3] M. Ahmadzadeh, D. Elliman, and C. Higgins. An Analysis of Patterns of Debugging Among Novice Computer Science Students. *SIGCSE Bull.*, 37(3):84–88, June 2005.

[4] J. Bennedsen and M. E. Caspersen. Failure rates in introductory programming. *SIGCSE Bull.*, 39(2):32–36, June 2007.

[5] N. C. C. Brown, M. Kölling, D. McCall, and I. Utting. Blackbox: A large scale repository of novice programmers' activity. In *Proc. SIGCSE '14*, SIGCSE '14, pages 223–228, New York, NY, USA, 2014. ACM.

[6] P. Denny, A. Luxton-Reilly, and D. Carpenter. Enhancing syntax error messages appears ineffectual. In *Proc. ITiCSE '14*, ITiCSE '14, pages 273–278, New York, NY, USA, 2014. ACM.

[7] P. Denny, A. Luxton-Reilly, and E. Tempero. All syntax errors are not equal. In *Proc. ITiCSE '12*, ITiCSE '12, pages 75–80, New York, NY, USA, 2012. ACM.

[8] 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 *Proc. ICER '09*, ICER '09, pages 3–14, New York, NY, USA, 2009. ACM.

[9] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas, and C. Zander. Debugging: Finding, Fixing and Flailing, a Multi-institutional Study of Novice Debuggers. *Computer Science Education*, 18(2):93–116, 2008.

[10] D. Hovemeyer, M. Hertz, P. Denny, J. Spacco, A. Papancea, J. Stamper, and K. Rivers. Cloudcoder: building a community for creating, assigning, evaluating and sharing programming exercises (abstract only). In *Proc. SIGCSE '13*, SIGCSE '13, pages 742–742, New York, NY, USA, 2013. ACM.

[11] M. C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proc. ICER '06*, ICER '06, pages 73–84, New York, NY, USA, 2006. ACM.

[12] S. K. Kummerfeld and J. Kay. The Neglected Battle Fields of Syntax Errors. In *Proc. ACE '03*, ACE '03, pages 105–111, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[13] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander. Debugging: The Good, the Bad, and the Quirky – a Qualitative Analysis of Novices' Strategies. *SIGCSE Bull.*, 40(1):163–167, Mar. 2008.

[14] C. Norris, F. Barry, J. B. Fenwick Jr., K. Reid, and J. Rountree. Clockit: collecting quantitative data on how beginning software developers really work. In *Proc. ITiCSE '08*, ITiCSE '08, pages 37–41, New York, NY, USA, 2008. ACM.

[15] A. Papancea, J. Spacco, and D. Hovemeyer. An open platform for managing short programming exercises. In *Proc. ICER '13*, ICER '13, pages 47–52. ACM.

[16] N. Parlante. Nifty Reflections. *SIGCSE Bull.*, 39(2):25–26, June 2007.

[17] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards improving programming habits to create better computer science course outcomes. In *Proc. ITiCSE '13*, ITiCSE '13, pages 243–248, 2013. ACM.