CodeWorkout: Short Programming Exercises with Built-in Data Collection

Stephen H. Edwards and Krishnan Panamalai Murali
Virginia Tech
Blacksburg, VA 24060
edwards@cs.vt.edu, pmk1433@vt.edu

ABSTRACT

Learning programming techniques can be challenging and frustrating for many students. Many instructors use drill-and-practice strategies to help students develop basic programming techniques and improve their confidence. Online systems that provide short programming exercises with immediate, automated feedback are often seen as a valuable approach to drill-and-practice. However, the relationship between practicing with short programming exercises and performance on larger programming assignments or exams is unclear. This paper describes CodeWorkout, an open-source drill-and-practice system that supports short programming exercises and multiple choice questions. CodeWorkout combines an open, gradual engagement model that allows any student to practice exercises, whether or not they have an account or are enrolled in a course, together with powerful course management features that support graded assignments, quizzes, and even practicum-style exams. It also provides a rich data collection and evaluation infrastructure for educational research purposes. We report on initial experiences using CodeWorkout in a CS1 course, including student perceptions of the tool and its benefits.

Keywords

CodeWorkout; homework; coding; skill development; practice; exam

1. INTRODUCTION

Rudimentary programming skills are essential to developing fundamental proficiency in computer science. But many computer science students lack effective resources to practice basic programming skills. Practicing manageably small programming exercises helps students gain confidence and hone their skills [3]. However, current programming practice systems provide minimal assistance to students struggling with exercises [3, 11]. Meanwhile, instructors would also benefit from a robust practice system that does not overburden them with heavy time and administrative demands.

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. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ITiCSE '17, July 03-05, 2017, Bologna, Italy

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4704-4/17/07...15.00

DOI: <http://dx.doi.org/10.1145/3059009.3059055>

A number of drill-and-practice tools are emerging to help students develop basic programming techniques and improve their confidence. However, these tools continue to grapple with trade-offs between adding burdensome workload on instructors, and sacrificing content quality [5]. This paper describes CodeWorkout, a new drill-and-practice system designed to build on previous work and provide a more comprehensive solution to small-scale practice assignments in the contexts of both individual learning, and learning in the CS classroom. CodeWorkout provides open programming practice to anyone, without requiring an account, but also provides for more structured, course-based use by instructors and students working with graded assignments. It also offers the facility for instructors or students to write their own questions, and provides a rich data model for capturing all data regarding exercise attempts, errors, and feedback. In addition to providing a basis for educational research, this data model also offers the opportunity to develop analyses based on item response theory for evaluating questions and quantifying student understanding, both to track student progress and to refine and improve exercises. In this paper we report on our initial experiences using CodeWorkout for both homework and on-line exam questions in a CS1 course, and describe student perceptions of use of the system.

Section 2 discusses related work, while Section 3 describes CodeWorkout and its unique features. Section 4 follows by describing how CodeWorkout was used in our course, then presents a summary of results from a student survey intended to collect subjective views of using the tool. Finally, Section 5 summarizes the lessons learned and describes future directions for CodeWorkout.

2. RELATED WORK

Drill-and-practice is a long-established method for rehearsing and exercising new knowledge. It is widely considered an effective technique for learning and is best implemented when accompanied by indicators of progress made [1]. Intelligent Tutoring Systems (ITS) typically employ cognitive models to track student progress and have demonstrated some effectiveness by aiding students with an artificial tutor [8, 9]. However, the artificial intelligence (AI) infrastructure of an ITS is non-trivial and developing and maintaining an AI tutor to support course-wide material on programming would be particularly challenging. Furthermore, Weber and Brusilovsky found that teachers preferred versatility of instructional technology over other special features [15].

More versatility can be found in a variety of online practice systems. Practice systems generally categorize questions

so that as students answer them, the system can track the students' progress within the category. However, there are practice systems with support for different types of questions and with other unique features. In the following sections, we highlight systems with notable approaches to learning.

2.1 Coding Systems

Coding systems have begun to emerge specifically to support practice at programming. Instead of questions, these systems ask students to solve problem descriptions through code. CodingBat (formerly JavaBat) offers a collection of small programming problems and now also supports instructor-contributed problems [11]. The accessibility of an open, online repository of practice exercises makes CodingBat a valuable resource for novice programmers. CodingBat takes advantage of test cases to evaluate the correctness of students' code. CodeWrite [3] similarly evaluates student code, but specifically holds students responsible for writing exercises and their respective test cases.

To explore the viability of a "student-sourced" approach to creating practice question banks, the 2015 study by Denny, Cukierman and Bhaskar [2] performed an empirical validation of the efficacy of student-created programming exercises in boosting learning. The study involved a set of randomly selected students who had to create a set of programming questions before a summative exam while the rest of the class did not have to. The study found that students who partake in the activity of creating viable practice programming questions achieve better grades. It was also shown that students were capable of producing exercises across a wide range of topics and difficulties.

WeBWork-JAG [5] is a practice system that uniquely extends a rich MCQ and FR system to also support coding exercises. Like several others, it allows students to provide questions. While they observed similar benefits of drill-and-practice as others, Gotel, et al. identified several challenges to practice systems. Most notably, they expressed concerns about the quality of student-provided content and the consequential demands of maintenance, exposing the need to address exercise quality in practice systems.

In addition to CodingBat, a related system that has also influenced CodeWorkout is CloudCoder [10], another open-source web-based system that is free to use for institutions. CloudCoder allows sharing of questions between instructors, and also supports the logging of keystroke-level data for research purposes. Unlike CodingBat, CloudCoder supports whole-program exercises, where the student code reads from the standard input and writes the computation result back into the standard output stream, in addition to simple function-based exercises, where the student completes a single method that returns a specific value.

Infandango [6] is a simple drill-and-practice system with a testing infrastructure that is similar to that of CodeWorkout. In this system, the final Java programs for an exercise are submitted by students through the web front-end after authentication. Then Infandango's JUnit daemon compiles the code, executes it in a sandboxed environment and runs the test cases against it. Students are then given feedback based on which cases passed and failed. The authors point out that the modularity of the system in the form of its authentication, web front-end and database are its unique selling points. However, Infandango relies on CoSign for authentication, an approach that is not used at many edu-

cational institutions, and one that makes it difficult to integrate with learning management systems. Infandango is also constrained by the fact that it works only for Java and not other programming languages.

2.2 Automated Grading Systems

Wilcox [16] provides a summary of the use of automation in undergraduate level CS courses, stating that are many benefits to using autograding, primarily resource savings. One benefit of automated approaches is that they enable instructors to assign multiple smaller questions, many more than would be practical with manual grading. Wilcox states that autograders impede neither the interest nor the academic performance of the students, but instead increase them if used appropriately.

Likewise, there are academic tools that take advantage of automated testing to assess program quality. Web-CAT [4] and Marmoset [13] are popular automated grading tools for computer science. Both use hidden unit tests—written by instructors—to test the correctness of students' programs. While automated graders may save instructors substantial time in grading programs, their strengths are in assessing student work. The design of CodeWorkout's infrastructure takes advantage of lessons learned from automated graders, but conversely focuses on facilitating learning over assessing larger-scale assignments.

3. CODEWORKOUT: A NEW DRILL-AND-PRACTICE SYSTEM

CodeWorkout is a completely online and open system for all those who are interested in teaching programming to their students. Thus, CodeWorkout is not limited to any particular institution. It provides a universal platform on which students from various backgrounds can practice programming and instructors can offer courses. CodeWorkout is not limited to short, single-method programming questions but is capable of supporting different kinds of questions, including multiple choice (both forced choice and multiple answer), coding by filling in the blanks, using arbitrary objects (lists, maps, or even instructor-defined classes) instead of only primitives, writing collections of methods or an entire class (instead of just single methods), multi-part questions that include multiple prompts, and "find and fix the bug" style questions where students are given a code implementation containing one or more errors to repair.

Figure 1 shows the view students use to enter answers for and receive feedback on exercises (video demo: <http://codeworkout.org/example.mp4>). CodeWorkout uses an "exercising" metaphor to reinforce the idea that users are building their strengths (at programming). As a result, it uses terminology based in this metaphor, calling individual problems exercises, while collections of problems are called workouts, and the public practice area is known as the "gym". Students who practice on their own earn experience points instead of a grade, and practice counts towards the system's notion of topic mastery, whether conducted on one's own or as part of a class assignment.

CodeWorkout facilitates active learning through both its drill-and-practice approach and the facility for students to create their own questions that they share for the benefit of their peers. CodeWorkout also provides a facility for instructors to share their programming exercises with other

[Home](#) / [Gym](#)

X1: Factorial 1

Write a function in Java called `factorial()` that will take a positive integer as input and returns its factorial as output.

```

1 public int factorial(int n)
2 {
3     if (n == 0)
4         return 0;
5     if (n == 1)
6         return 1;
7     return n * factorial(n - 1);
8 }
9

```

[Check my answer!](#)

Feedback

Behavior	Result
factorial(0) <i>Remember that zero factorial is one</i>	
factorial(1) -> 1	
factorial(2) -> 2	
factorial(3) -> 6	
hidden	
hidden	

Figure 1: CodeWorkout's exercise interface.

instructors if they wish. It has the feature of tagging each question/assignment with all the concepts involved in them. In addition to supporting search features and an organic classification scheme, this also allows the system to track the scores of students on a concept-by-concept basis.

CodeWorkout aims to be a comprehensive system that addresses many of the weaknesses of its predecessors while meeting a wide variety of real classroom needs. As a result, CodeWorkout's design targets the following objectives:

- **Simultaneously supporting courses and the gym:**

CodeWorkout is designed to fully support classroom use by instructors who wish to use graded assignments. At the same time, CodeWorkout also provides a completely open “free practice” area where anyone, whether enrolled in a course or not, can browse and practice a large collection of publicly available exercises—a concept successfully pioneered by CodingBat. CodeWorkout provides full support for both uses. Classroom activities can include graded or optional homework, untimed or timed quizzes, or even live exams. Several workout policies are available so instructors can choose whether or not students can see scores or full feedback immediately (in homework) or not (during a test), and whether questions can be immediately reviewed, or review must wait until all students have completed. For timed quizzes and tests, CodeWorkout even allows instructors to set individual deadlines or allow extra time for specific students where necessary, to accommodate make-up work or students with special needs.

- **Gradual engagement:** CodeWorkout's open practice features are designed around the idea of *gradual engagement*—that is, users can use many of the site's features without an account, and they are not required to login or create an account until they reach the point where identifying information is required—such as enrolling in a course or needing to complete a course-specific assignment. Long-term sessions are used to track users, including anonymous users, so no data is

lost due to session timeouts or browser restarts. Once users reach a point where an account is required, they are prompted to provide the minimum necessary information: providing an e-mail address or logging in using Gmail or Facebook, for example. By providing a low effort login process that is delayed as much as possible, new users have a smoother entry path into the system. By using long-term sessions, once a user logs in all of their anonymous activities can be carried over into their account without loss of information.

- **Role-based permissions model:** CodeWorkout uses an egalitarian approach to security. Except for system administrators, all users are equal. Permissions are governed by one's role with respect to a specific piece of data. This idea is not novel, but is rare in many educational systems. Role-based permissions ensure that one who is listed as an instructor for a course has instructor-level access within that course, but nowhere else, for example. As a result, CodeWorkout is set up so that anyone can create a course, choosing to allow students to self-enroll or to upload a roster. Since self-proclaimed instructors only have access to their own courses, a more open and accommodating environment with lower barriers to entry can be maintained.

- **Exercise organization:** Exercises are the main focus of any drill-and-practice system and CodeWorkout aims to provide significant support features and facilities organized around it. Exercises are available to be either directly used in the gym or to be organized into workouts. The exercise model is truly polymorphic in CodeWorkout: an exercise can be of different types like multiple-choice or coding; it can also consist of multiple parts, allowing for a richer variety of questions. Questions are automatically versioned internally, so that the prompt a student is working on can be separated from edits being made concurrently by the question's author, and so that collected data on performance and answers can be tracked properly

through a question's edit history. Families of related questions are linked so that instructors can easily find alternative versions of a question, or avoid putting alternative versions of a question on a single assignment.

- **Collaborative and active learning:** CodeWorkout allows instructors to share exercises with one another. Its framework facilitates students who have mastered a particular topic to be able to create their own programming questions that they can share for the practice of their peers. Finally, it also tries to provide students with a facility to share hints for exercises and to participate in a Stack Overflow-like question forum associated with each question.
- **IRT statistics:** CodeWorkout aims to support the collection of statistics necessary to analyze questions and student performance using item response theory [14]. Using the workout scores as a basis for student skill level, CodeWorkout can determine the difficulty and discrimination values for student-created content like exercises and hints. As the system collects more and more data, these values can be used to filter out exercises and hints that are not as useful.

While CodeWorkout is still quite young, it aims to fill gaps in current support for small drill-and-practice coding problems. Providing classroom support that spans the range from open, unstructured (and even unenrolled) voluntary practice, all the way to in-class timed exam support, is an ambitious goal. It is also important to strive for supporting a broader audience of users. In their survey paper of automatic assessment systems, Ihantola et al. [7] note that most autograding systems have not been widely adopted because they were created solely for the sake of a thesis or a paper publication. They do not focus on building features that would make them a viable system for continuous use outside their parent institution. They point out significant barriers to wide-scale adoption. Vreda Pieterse also describes reasons that inhibit success of such systems [12]. She notes that setting up courses is problematic because they are not free, require extensive work for deployment on an institution's server, or suffer from significant usability issues. For these reasons, a robust solution targeted explicitly for external adoption is necessary.

4. STUDENT PERCEPTIONS

CodeWorkout has been used in two to three courses each semester at Virginia Tech during the 2015-2016 and 2016-2017 academic years, including use by 372 students in a CS1 course during Fall 2015, and 378 students in the same course in Spring 2016. CodeWorkout was used for graded homework assignments, for optional practice assignments, and for coding questions on in-class proctored and timed examinations. Students also had larger program assignments (graded using Web-CAT), as well as homework assignments that did not involve programming in this course. Finally, many students elected to voluntarily complete additional questions on CodeWorkout in a free practice mode, in order to study or prepare for exams.

While instructors reacted strongly positively to CodeWorkout as a classroom tool, we were also interested in how students reacted to the system. We distributed an opinion survey to 378 students in the spring 2016 CS1 course

just before students took Test 2. At that point, these students had already been using CodeWorkout for both voluntary and graded assignments, including in-class tests. The survey included nineteen questions, sixteen of which were five-point Likert scale questions rated from Strongly Agree to Strongly Disagree. In order to ascertain the consistency of results, these statements were both negatively and positively phrased. These statements deal with various topics pertaining to how well CodeWorkout was meeting its objectives, such as the usability of its interface, ease of access and usefulness of feedback. The final three questions were free-form text answer questions intended to gauge initial impressions, perceived drawbacks and overall usefulness of CodeWorkout. 173 students responded to the survey.

4.1 Likert-scale Question Results

Table 1 lists the mean scores and variances for the Likert-scale questions. Overall, students have a positive impression of CodeWorkout and see it as beneficial. The high means for questions 1, 2, 6, 8, 9, 10 and 15 were especially notable. These questions dealt with the usability and ease of access of CodeWorkout, two of its primary goals. Responses to two of the negative questions, Q3 and Q12, further support this, with low mean responses of 2.39 and 2.21 respectively. The high mean scores of 4.18 in Q8 (regarding the continuous availability of CodeWorkout) and 4.02 in Q15 (regarding ease of course enrollment) is an affirmation of this. The highest mean of all (4.31 in Q6) indicated that CodeWorkout was better than traditional paper-based settings for coding.

The only questions that were not as strongly supportive were Q4 and Q5. Q4 is a negatively phrased question asking how slow is CodeWorkout's feedback. The response to this has a mean of 2.85 (lower is better). While this certainly indicated that most students did not feel that the system was slow, a minority of students did feel that its speed can be improved. The positively phrased question with the lowest mean was Q5, which was directed towards a specific feature of CodeWorkout that allows students view their work in past assignments. While the mean of 3.57 indicates that the feature works, it also conveys that we could do more to improve it.

A majority of the respondents for Q1 (94.2%) said that they either agreed or strongly agreed to the statement that it was easy to submit exercise solutions in CodeWorkout. CodeWorkout's coding interface has received special praise: more than 85% of the respondents said that it was easy to write code with it.

Even though CodeWorkout is still very much under development, a majority of the students (63.9%) expressed their wish to use CodeWorkout in their next programming course. In Q9, 81.9% of students either agreed or strongly agreed that CodeWorkout is useful for learning introductory programming skills, its most important objective. Putting it a little differently, Q11 asks how much CodeWorkout has improved their programming skills. More than 76% of the responding students indicated that they either agreed or strongly agreed with this statement. Finally, with regard to usefulness in a traditional classroom setting, 78.6% of the respondents either agreed or strongly agreed that that the practice afforded by CodeWorkout will positively impact their performance in this course. These results present evidence that students see CodeWorkout as a viable drill-and-practice system in introductory programming settings.

Table 1: Student Survey Results

Question		Mean	Variance
<i>Usefulness</i>			
9	CodeWorkout is helpful to use for learning introductory programming	3.99	0.92
11	The programming practice I get from CodeWorkout has improved my programming skills	3.83	0.88
13	I believe that practicing on CodeWorkout will positively impact my performance on exams in this course, and on my grade in this course	3.85	0.8
14	I wish to use CodeWorkout in my next programming course	3.67	1.12
<i>Usability</i>			
1	It was easy for me to submit my solutions to exercises on CodeWorkout	4.29	0.47
6	Practicing and submitting exercises on CodeWorkout is simpler as compared to paper-based systems	4.31	0.60
10	I find CodeWorkout's coding interface easy to use	4.10	0.53
12	I found it difficult to write my solutions for the exercises on CodeWorkout because of its interface‡	2.21	0.76
<i>Access and navigability</i>			
2	It was simple for me to login to CodeWorkout for the first time	4.08	0.75
3	I was difficult for me to navigate to the exact course content that I was looking for on CodeWorkout‡	2.39	1.09
5	I was able to access my answers to previous assignments and exercises without complications	3.57	1.09
8	CodeWorkout is always available and accessible to me	4.00	0.50
15	The software made finding and opening my past programs easy.	4.02	0.59
<i>Feedback</i>			
4	The automatic feedback generated by CodeWorkout was slow‡	2.85	0.96
7	The compilation error messages that CodeWorkout gives are useful for fixing syntax errors in my code	3.44	1.22

4.2 Free-form Question Results

The first free-form text question was: “What were your initial impressions about CodeWorkout as a programming practice system?” 168 of the total 174 respondents answered this question. From the student responses to this question, we can see that a majority of students had a positive first experience. The following are some of the sample responses:

- *It was helpful to see the tests ran against my code to see where I went wrong. The error messages also help me find the faults in my code*
- *Simple and gave feedback (which is invaluable).*
- *I like how it feels like an online game. Something that makes me want to achieve more and fill out the progress bar.*
- *It was fun to be given a problem, develop a solution, and immediately receive feedback about the task I completed, which was a welcome change from the needed practice in other formats in the course.*

The second free-form question was: “What were the aspects of CodeWorkout that you found to be an impediment to your learning?” Students complained about insufficient feedback when their solution did not work correctly. This is understandable because most of the students were first time Java programmers who may find compiler error messages to be cryptic. Also students sometimes were unable to figure out why their code was repeatedly failing some test cases. Unfortunately, this situation arose because hints and

feedback at the test case level were not specified for the exercises used in this study due to lack of time, *even though CodeWorkout provides this capability.*

From the second free-form question, we understand that students had some difficulties in working out the computational logic for individual exercises and they were frustrated that CodeWorkout couldn't help them out when they were stuck. This was not an impediment *per se* to learning, but is an interesting direction to feedback that we did not consider before. Overall, the responses to this question were rather positive and this reinforces the inference made from the Likert-scale questions that CodeWorkout does not pose any barriers to student learning.

The final free-form question in the survey was: “Overall, how useful do you think CodeWorkout is to your learning and improvement of programming skills?” Of all three free-form text questions in the survey, the responses to this question were the most favorable. The few negative replies to this question pointed out the lack of certain features like more descriptive feedback or display of alternative working solutions. We would like to point out that these are features we are already building into CodeWorkout for next semester. The following are some of the sample responses:

- *Useful for studying.*
- *I think it's been quite useful for learning*
- *CodeWorkout hasn't helped me much in terms of improving my programming skills. It is good practice, and if hadn't had prior programming experience when using this website I would have benefited much more from it.*

- It gives an opportunity to practice standard coding skills in a non graphical context. Personally I think that exposure to coding in this manner will be much more reflective of future/actual coding we will do.

5. CONCLUSION

CodeWorkout is a new drill-and-practice system designed to provide a larger range of opportunities for students to practice basic coding skills. Inspired by predecessors, it design aims to combine the best strengths of prior work with new strategies for enhancing classroom support and supporting student practice. In the experiences reported in this paper, CodeWorkout was used quite successfully in an introductory course with a large number of students. Students reacted positively and appear to see clear benefits to using this style of exercise practice to develop their skills.

Nevertheless, there are still many research questions needing further exploration. In the future, we plan to continue refining CodeWorkout to support such investigations. An important part of this effort is the use of item response theory to characterize the performance of questions and the identification of questions that may need revision or editorial attention. At the same time, IRT offers a deeper, more data-driven way of estimating student ability. By modeling both student ability and question performance, it is possible to intelligently recommend new problems for practice that are closer to the student's zone of proximal development. The addition of social features and Stack Overflow-inspired Q&A discussions for questions offers a unique strategy to try to help students who get stuck. Similarly, prompting successful students to offer hints on how to tackle questions they have completed, together with analysis of when those hints help later students get unstuck, offer new strategies for engaging students in the community of users, instead of encouraging them to pursue individual practice in isolation. A consolidated approach to these issues is more likely to meet the needs of current and future educators.

6. ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under grant DUE-1245589. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

- [1] J. D. Bransford, A. L. Brown, R. R. Cocking, et al. How people learn, 2000.
- [2] P. Denny, D. Cukierman, and J. Bhaskar. Measuring the effect of inventing practice exercises on learning in an introductory programming course. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, Koli Calling '15, pages 13–22, New York, NY, USA, 2015. ACM.
- [3] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx. Codewrite: Supporting student-driven practice of java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476, New York, NY, USA, 2011. ACM.
- [4] S. Edwards. Web-CAT. <http://web-cat.org>. last accessed 04-07-2016.
- [5] O. Gotel, C. Scharff, and A. Wildenberg. Extending and contributing to an open source web-based system for the assessment of programming problems. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*, PPPJ '07, pages 3–12, New York, NY, USA, 2007. ACM.
- [6] M. J. Hull, D. Powell, and E. Klein. Infandango: Automated grading for student programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 330–330, New York, NY, USA, 2011. ACM.
- [7] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. ACM.
- [8] K. R. Koedinger and E. L. F. Sueker. Pat goes to college: Evaluating a cognitive tutor for developmental mathematics. In *Proceedings of the 1996 International Conference on Learning Sciences*, ICLS '96, pages 180–187. International Society of the Learning Sciences, 1996.
- [9] P. D. Palma. Viewpoint: Why women avoid computer science. *Commun. ACM*, 44(6):27–30, June 2001.
- [10] A. Papancea, J. Spacco, and D. Hovemeyer. An open platform for managing short programming exercises. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 47–52, New York, NY, USA, 2013. ACM.
- [11] N. Parlante. CodingBat. <http://codingbat.com/>. last accessed 04-06-2016.
- [12] V. Pieterse. Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, CSERC '13, pages 4:45–4:56, Open Univ., Heerlen, The Netherlands, The Netherlands, 2013. Open Universiteit, Heerlen.
- [13] J. Spacco. Marmoset. <http://marmoset.cs.umd.edu/>. last accessed 04-16-2016.
- [14] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
- [15] G. Weber and P. Brusilovsky. Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education (IJAIED)*, 12:351–384, 2001.
- [16] C. Wilcox. The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 90–95, New York, NY, USA, 2015. ACM.