PCEX: Interactive Program Construction Examples for Learning Programming

Roya Hosseini
University of Pittsburgh
Pittsburgh, PA
roh38@pitt.edu

Kamil Akhuseyinoglu
University of Pittsburgh
Pittsburgh, PA
kaa108@pitt.edu

Andrew Petersen
University of Toronto Mississauga
Mississauga, Canada
andrew.petersen@utoronto.ca

Christian D. Schunn
University of Pittsburgh
Pittsburgh, PA
schunn@pitt.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, PA
peterb@pitt.edu

ABSTRACT

A sizable body of research on instructional practices supports the use of worked examples for acquiring cognitive skills in domains such as mathematics and physics. Although examples are also important in the domain of programming, existing research on programming examples is limited. Program examples are used by instructors to achieve two important goals: to explain program behavior and to demonstrate program construction patterns. Program behavior examples are used to demonstrate the semantics of various program constructs (i.e., what is happening inside a program or an algorithm when it is executed). Program construction examples illustrate how to construct a program that achieves a specific purpose. While both functions of program examples are important for learning, most of the example-focused research in computer science education focused on technologies for augmenting program behavior examples such as program visualization, tracing tables, etc. In contrast, advanced technologies for presenting program construction examples were rarely explored. This work introduces interactive Program Construction Examples (PCEX) to begin a systematic exploration of worked-out program construction examples in the domain of computer science education. A classroom evaluation and analysis of the survey data demonstrated that the usage of PCEX examples is associated with better student's learning and performance.

CCS CONCEPTS

• **Social and professional topics → Computing education**; **CS1**; • **Applied computing → Interactive learning environments**; *E-learning*; *Computer-assisted instruction*;

KEYWORDS

worked examples, CS1, computer science education, Java programming, interactive learning technologies

ACM Reference Format:
Roya Hosseini, Kamil Akhuseyinoglu, Andrew Petersen, Christian D. Schunn, and Peter Brusilovsky. 2018. PCEX: Interactive Program Construction Examples for Learning Programming. In *18th Koli Calling International Conference on Computing Education Research (Koli Calling '18), November 22–25, 2018, Koli, Finland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3279720.3279726

1 INTRODUCTION

Program code examples play a crucial role in learning how to program. Instructors use examples extensively to demonstrate the semantics of the programming language being taught, to introduce problem-solving approaches, and to highlight fundamental coding patterns. Programming textbooks also devote attention to examples with a large proportion of textbook space consumed by program examples and associated comments. Moreover, the code of all presented examples is typically provided in accompanying materials to encourage students to explore, run, and modify the examples. In contrast, the work on e-learning tools for computer science education uses code examples quite unevenly. While there is an active stream of work focused on *code behavior examples* (usually presented as dynamic code animations), there are very few e-learning tools focused on *program construction* examples, i.e., a step-by-step demonstration on how to solve a specific programming problem. These examples are typically presented online in a *passive* learning form - as a static code or, rarely, as a screencast. In this aspect, computer science education significantly lags behind other areas such as mathematics and physics where learning technologies for presenting problem-solving examples have been extensively studied [1]. This paper attempts to bridge this gap. We introduce PCEX, an online learning tool focused on introducing program construction examples to students. PCEX presents examples in an interactive, engaging form in order to increase students' motivation to work with examples and improve their learning. We also report the results of a semester-long exploratory evaluation of PCEX in an introductory programming class as well as the survey that was carried out to understand the perception of the students about the tool. Our results demonstrated interesting data about

Koli Calling '18, November 22–25, 2018, Koli, Finland
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6535-2/18/11…$15.00
https://doi.org/10.1145/3279720.3279726

the relationships between the use of PCEX examples and student's learning.

2 RELATED WORK

2.1 Worked Examples in Programming

To make the message of this paper clearer, we classify program examples that have been used in teaching and learning programming into two groups according to their primary instructional goal: program behavior examples and program construction examples. Program behavior examples are used to demonstrate the semantics (i.e., behavior) of various program constructs (i.e., what is happening inside a program or an algorithm when it is executed). Program construction examples attempt to communicate important programming patterns and practices by demonstrating how a program that achieves various meaningful purposes (e.g., summing an array) is constructed. This distinction might not be clearcut for examples with no augmentation since the same example code be used for both purposes. However, the attempts to augment examples with learning technologies to increase their instructional value (i.e., add code animation or explanations) usually focus on one of these goals.

Program behavior examples have been extensively studied. While in textbooks and tutorials program behavior is still explained by using textual comments attached to program code, a more advanced method for this purpose - *program visualization* that visually illustrates the runtime behavior of computer programs - is becoming increasingly more popular. Over the past three decades, a number of specialized educational tools for observing and exploring program execution in a visual form have been built and assessed [26]. The decades of research in this area have demonstrated that animated examples have the highest impact when students are *interactively engaged* in the work with examples rather than watching the animations passively. Several variants for engaging students have been explored such as making animations more interactive (allowing students to explore forwards and backwards or to enter their own data), introducing challenges (asking students to predict the next step), and asking students to construct the animations themselves [21, 26, 27].

In contrast to interactive and engaging program behavior examples, program code examples are typically presented online as text with comments [20] or as video fragments with instructor narration over slides or a code editor window [25]. Such passive presentation does not allow for exploration and engagement. In our work on PCEX, we attempted to use the research findings in the area of program behavior examples to produce interactive program construction examples that better engage students and improve their learning.

2.2 Program Construction Assessment Tools

Program construction assessment tools are currently the most popular e-learning technology to help students in acquiring program construction knowledge. Early programming assessment tools received uploads of entire student programs and evaluated them against a set of instructor-defined tests [2]. A popular example of these assignment-focused tools is Web-CAT [9]. More recent tools have evolved into Web sites which pose program construction problems to students and allow them to submit their code through a Web form. Many of these tools ask students to write small pieces of code, rather than entire programs, and offer "skeleton" code as a starter [18]. Nick Parlante's CodingBat is one of the earliest examples of these tools [23]. This model has been adopted by several tools, including CodeWrite [8]; CodeAssessor [29]; PCRS [30]; CloudCoder [15]; and CodeWorkout [3], which was developed from the Web-CAT project.

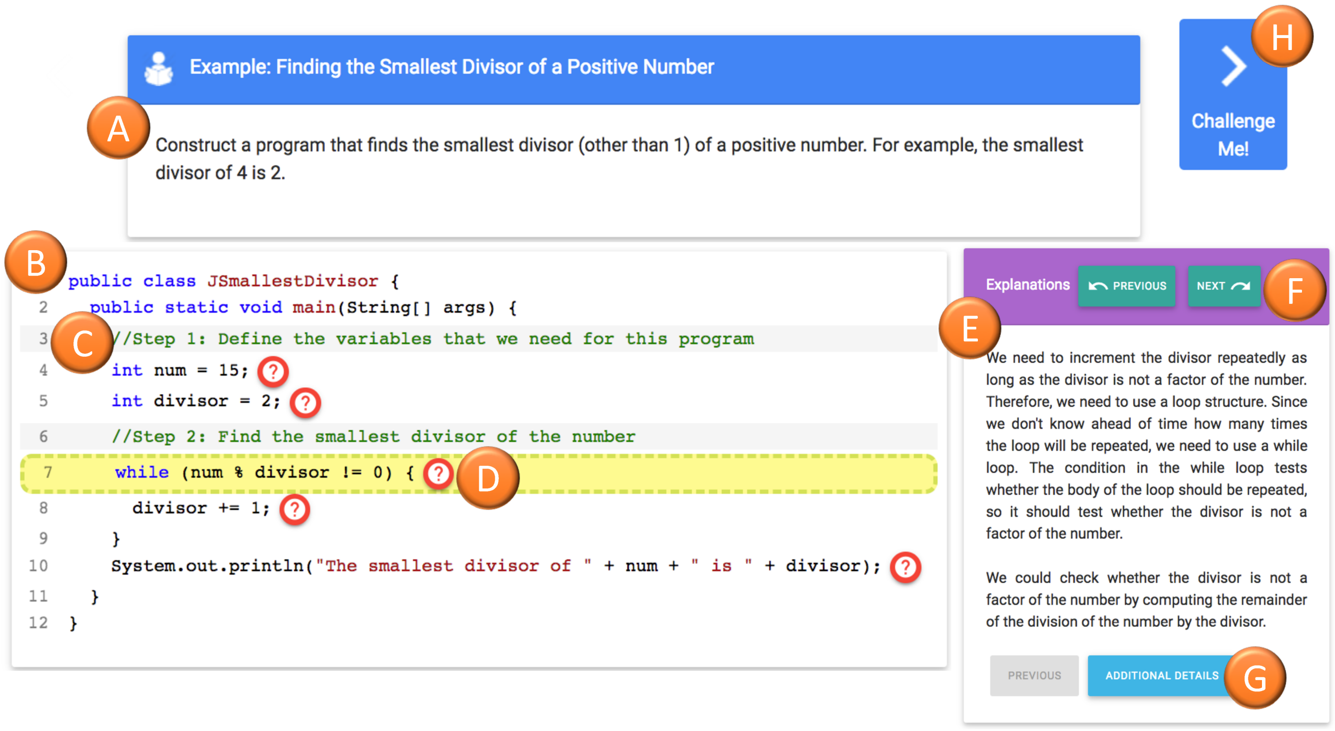
2.3 Integrated Systems

While animated examples and program construction assessment tools were originally designed as independent systems, platforms that incorporate more than one type of these tools have become increasingly popular in "inverted courses", MOOCs, ebooks [4, 5, 7, 10], and online practice systems [11, 14]. As a result, a number of recent tools were designed to be easily re-usable in different contexts. For example, the Online Python Tutor (OPT) [12], which provides memory visualizations for a range of languages, has been incorporated into ebooks [10], MOOCs and online courses [12]. Our work follows this approach. PCEX examples were designed as re-usable learning content. In our studies, the access to examples was provided through an integrated practice system, which also offered students programming problems served by a program construction assessment tool.

3 PCEX: CHARACTERISTICS AND DESIGN

PCEX (Program Construction EXamples) is an interactive tool to support mastering program construction skills through examples. The innovative idea behind PCEX is to create "rich examples" that support free exploration and challenge the student. Figure 1 illustrates a PCEX example. Each PCEX example includes a "goal" (Figure 1, A) and worked program steps (Figure 1, B). The goal states what function the example program performs. The worked steps begin with a subgoal label (Figure 1, C) and are represented in the form of sequence of short fragments of code (no more than a few lines of code) that illustrate how the program is constructed. Labeling subgoals in worked examples is known to increase student performance by leading students to group a set of steps and encouraging them to self-explain the reason for clustering those steps [6]. The example is enriched with instructional explanations that are shown as question mark icons next to all or a subset of example lines (Figure 1, D). Once a student clicks on a question mark, an explanation is shown on the right side (Figure 1, E). The student can request additional details for the selected line by clicking on the "Additional Details" button (Figure 1, G) or can navigate to the previous or next line to read an explanation (Figure 1, F).

In addition to being *explorable*, PCEX examples *challenge* students by engaging them into a problem-solving activity. When a student clicks on the "Challenge me" button (Figure 1, H), an interactive challenge activity is presented to the student as shown in Figure 2. The goal of a challenge is to encourage students *to apply* the program construction knowledge presented in the original example to *self-assess* whether their understanding is correct. In essence, a challenge is a programming problem that is very similar to the original example in both the goal to achieve and the code. A challenge has a problem statement (Figure 2, I) and code. However, the code has no explanation and is not complete – one or more of



Example: Finding the Smallest Divisor of a Positive Number

Construct a program that finds the smallest divisor (other than 1) of a positive number. For example, the smallest divisor of 4 is 2.

```
public class JSmallestDivisor {
  public static void main(String[] args) {
    //Step 1: Define the variables that we need for this program
    int num = 15;
    int divisor = 2;
    //Step 2: Find the smallest divisor of the number
    while (num % divisor != 0) {
      divisor += 1;
    }
    System.out.println("The smallest divisor of " + num + " is " + divisor);
  }
}
```

We need to increment the divisor repeatedly as long as the divisor is not a factor of the number. Therefore, we need to use a loop structure. Since we don't know ahead of time how many times the loop will be repeated, we need to use a while loop. The condition in the while loop tests whether the body of the loop should be repeated, so it should test whether the divisor is not a factor of the number.

We could check whether the divisor is not a factor of the number by computing the remainder of the division of the number by the divisor.

Figure 1: A worked example in the PCEX activity. The example includes the goal (A), interactive worked code (B), the subgoal label presented as a comment (C), the link to instructional explanations (question mark symbols) (D), explanations (E), a navigation link to the explanation for the previous/next line (F), additional details for the highlighted line (G), and a challenge navigation link (H).

the code lines are missing. The student's goal is to complete the code by dragging and dropping lines from the set of options (Figure 2, J) to each of the missing fields. This drag-and-drop interaction approach is similar to Parsons problems (puzzles) [24]. The student can check whether the challenge is solved correctly by clicking on the "Check" button (Figure 2, K). The feedback is presented to the student by highlighting correctly (in green) and incorrectly (in red) placed lines. The student can also request a hint or more detailed feedback (Figure 2, L). If the student cannot solve the challenge with three attempts, she can request the solution.

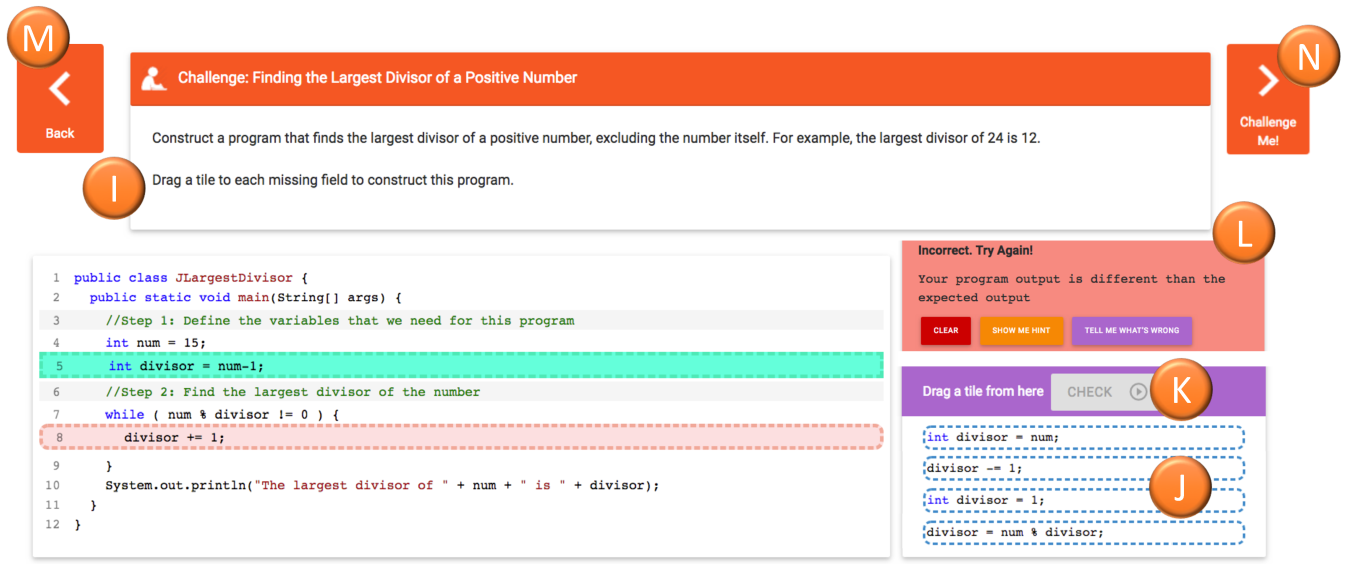
At any moment, the student can navigate to the core explained example (Figure 2, M). Also, if an example has several challenges, the student may navigate between them. The student can navigate to the next challenge only when the current challenge is solved or the solution is seen after the third incorrect attempt (Figure 2, N).

In this work, we use the term *PCEX activity* to refer to the PCEX worked example and its associated challenges. Each PCEX activity was created by annotating the example and challenge code with a set of predefined tags. The annotated code was parsed to generate a corresponding JSON file for each of the PCEX activities. The JSON file was used by a single-page JavaScript application to support interactive work with examples and challenges as shown in Figures 1 and 2. The current version of the PCEX supports any executable code in Java or Python. Note that the program code for a challenge requires to produce an output as PCEX employs an output-based evaluation of the student answer.

4 CLASSROOM EVALUATION STUDY

We conducted a classroom study to evaluate the relationship between using PCEX examples and students' learning of programming concepts. The subjects were students enrolled in an undergraduate Introductory Java Programming course in Fall 2017. The course had two sections with a shared syllabus and lecture materials. The students who took the course were not required to have any prior programming background.

The study followed a pre/post-test experimental design to examine the relationship between using PCEX examples and student's learning. In the beginning of the semester, students completed a pre-test consisting of six questions that evaluated knowledge of a subset of programming constructs covered in the course. The questions were designed to cover both easy and complex concepts in programming. The first three questions asked the student to complete code by filling in blank line(s) or writing small code snippets. The remaining three questions asked the student to determine the correct order of provided lines of code to achieve a certain purpose. The lines were shuffled and included distractors. A post-test isomorphic to the pre-test was administered at the end of the semester. Maximum possible score on the pre/post-test was 29: 14 points for the first three questions and 15 points for the last three questions.



Challenge: Finding the Largest Divisor of a Positive Number

Construct a program that finds the largest divisor of a positive number, excluding the number itself. For example, the largest divisor of 24 is 12.

Drag a tile to each missing field to construct this program.

```
public class JLargestDivisor {
  public static void main(String[] args) {
    //Step 1: Define the variables that we need for this program
    int num = 15;
     int divisor = num-1;
    //Step 2: Find the largest divisor of the number
    while ( num % divisor != 0 ) {
      divisor += 1;
    }
    System.out.println("The largest divisor of " + num + " is " + divisor);
  }
}
```

Incorrect. Try Again!

Your program output is different than the expected output

```
int divisor = num;
divisor -= 1;
int divisor = 1;
divisor = num % divisor;
```

Figure 2: A challenge in the PCEX activity that follows the worked example in Figure 1. The challenge includes the goal (I), has one or more missing lines, and asks the student to drag and drop a line from the given options (J) to each missing line to construct the program. A student can request feedback by pressing the "Check" button (K). The feedback message is shown in part (L). The student can go back to the example by pressing the "Back" button (M). If there are more challenges available, the student can go to the next challenge by pressing the "Challenge Me" button (N). This button is shown only when the current challenge is solved or the student checks the solution after the third incorrect attempt.

All students were provided with a link and an individual account to access a practice system that included 55 PCEX activities, as described in Section 3, and 46 coding exercises served by the PCRS tool [30]. The learning content was organized into 14 topics. All PCEX activities started with a worked example and were followed by 1–3 challenges (the median was 1). In total, PCEX activities included 55 interactive examples with 628 line explanations and 76 challenges. Although the use of the practice system was voluntary, students were encouraged to use the system by offering extra credit for those who completed at least 3 PCEX activities (i.e., viewed the examples and solved all the associated challenges for the example they viewed) and solved 7 coding problems. All the practice content in the system were accessed through Mastery Grids [19]. To engage students to work with the content, Mastery Grids provides visual personal progress tracking (known as Open Student Modeling (OSM)), as well as social comparison visualizations (known as Open Social Student Modeling (OSSM)). Only the OSM features of the interface were enabled for this study.

5 COLLECTED DATA

Data from students in all sections were combined. 71 students took the final exam. 64 took both the pre-test and post-test and logged in to the system, and 62 attempted at least one activity (i.e., loaded a *PCEX* activity or attempted a coding exercise). Table 1 shows the summary statistics for all usage variables, after removing outliers[1].

We measured the usage of activities by counting the number of examples accessed, example line clicked, challenges and coding exercises solved, and *PCEX* activities completed (that is, all challenges associated with the examples were solved). We also tracked the time spent on each of the challenges and activities. For the examples, we distinguished between the time a student spent inspecting the examples before clicking any of the lines and the time the student spent reading the explanation while clicking through example lines. The total time spent on examples is the sum of these two. The total time spent on *PCEX* activities includes the total time the student spent on worked examples and associated challenges.

On average, students accessed 31 (56%) worked examples, clicked on 72 (11%) example lines, solved 38 (50%) distinct challenges and 17 (37%) distinct coding exercises, and completed 28 (51%) *PCEX* activities. The average time that students spent inspecting the examples before clicking any of the lines with explanations (49 mins.) was about the same as the average time that students spent accessing explanations (47 mins.). The average time spent on challenges (80 mins.) was about 2.7 times less than the average time students spent on coding exercises (213 mins.). Overall, the average total time that students spent on *PCEX* activities (examples and challenges) was 177 mins., which is comparable to the time spent on coding exercises.

Many of these fine-grained measures were highly correlated with one another and should not be considered as independent measures. In particular, the number of distinct successful attempts

[1]We excluded the line clicks of one student who clicked on 400 (64%) example lines and also the time on examples for another student who spent over 500 minutes on examples. It should be noted that using all student data showed the same pattern of results for all analyses.

Table 1: Summary statistics for usage of *PCEX* and coding exercises by students who logged in to the system and attempted at least one activity (N=62).

	Median	Mean	Min	Max
EXAMPLES				
Example accesses	33	30.7	1	55
Example line clicks	41.5	72.4	0	517
Time on examples before line clicks (mins.)	29.5	49.3	0.4	273.8
Time on example lines (mins.)	26.2	47.0	0	302.3
Total time on examples (mins.)	60.3	96.4	0.4	576.1
CHALLENGES				
Challenge attempts	80	86.2	0	336
Challenges solved	40	41.7	0	96
Distinct challenge attempts	38.5	38.4	0	76
Distinct challenges solved	37	38.1	0	76
Time on challenges (mins.)	59.4	80.5	0	342
PCEX ACTIVITIES				
PCEX activities completed	27	27.8	0	55
Total time on *PCEX* activities (mins.)	115.9	176.9	0.4	918.1
CODING EXERCISES				
Coding exercise attempts	71	105.5	0	382
Coding exercises solved	12.5	19.3	0	73
Distinct coding exercise attempts	14	19.2	0	46
Distinct coding exercises solved	11	17.2	0	46
Time on coding exercises (mins.)	128.8	212.5	0	868.2

on *PCEX* challenges was highly correlated with challenge attempts ($\rho = 0.92$), challenges solved ($\rho = 0.99$), distinct challenge attempts ($\rho = 1$), and time on challenges ($\rho = 0.83$). Similarly, the number of distinct successful attempts on coding exercises was highly correlated with coding exercise attempts ($\rho = 0.88$), coding exercises solved ($\rho = 0.99$), distinct coding exercise attempts ($\rho = 0.99$), and time on coding exercises ($\rho = 0.9$). *PCEX* activities were also highly correlated to the total time on *PCEX* activities ($\rho = 0.74$). We also found a moderate correlation between example accesses and example line clicks ($\rho = 0.44$) and a strong correlation between example line clicks and time on example lines ($\rho = 0.87$).

After examining the correlations between these variables, we decided to use only three independent variables that were not highly correlated: example line clicks, representing the amount of interaction with examples; *PCEX* activities completed, representing the total work done with *PCEX* activities; and distinct coding exercises solved, representing the amount of work done on coding exercises.

6 RESULTS

We started by investigating the correlation between usage of *PCEX* activities and student's learning. This overall usage analysis is then complemented with a more detailed analysis that describes the relationship between usage of *PCEX* and student's learning over time and identifies which usage behaviors resulted in better learning.

6.1 Relationship between usage of PCEX and student's learning

We evaluated the correlation between usage of *PCEX* activities and student's learning, using several measures of process success and outcomes: (1) learning gain, which is defined as the ratio of the actual gain (post-test score minus pretest score) to the maximum possible gain (maximum achievable post-test score minus pretest score); (2) number of challenges that the student solved; (3) number of coding exercises that the student solved; (4) midterm grade; and (5) final exam grade.

6.1.1 Correlation between usage of PCEX and learning gain. Learning gain was calculated for the 64 students who had taken both pre-test and post-test, answering all of the questions in the test. The learning gain followed a normal distribution and ranged from 0.07 to 1.0 with a mean of 0.58. The number of example line clicks were not correlated with learning gain; however, *PCEX* activities completed had a significant positive correlation with learning gain ($\rho = 0.30, p = .02$). In addition, coding exercises were found to have a strong positive correlation with learning gain ($\rho = 0.62, p < .001$).

6.1.2 Correlation between usage of PCEX and performance in coding exercises. We looked into the relationship between usage of *PCEX* activities and distinct successful attempts on coding exercises and found that working with *PCEX* activities was positively correlated with student coding performance: the number of example line clicks ($\rho = 0.31, p = .01$) was correlated with distinct successful attempts on coding exercises, as was the number of *PCEX* activities completed ($\rho = 0.71, p < .001$).

We also ran multiple regression analyses to examine whether the *PCEX* activities and example line clicks could significantly predict the number of distinct coding exercises solved, controlling for prior knowledge, as measured by the pre-test. We fitted two multiple regressions, one with example lines clicked and pre-test score as factors and one with *PCEX* activities completed and the pre-test as factors. Both were significant independent predictors of distinct coding exercises solved even after controlling for the pre-test: each additional example line click and each *PCEX* activity a student completed resulted in a 0.05 ($SE = 0.02, p = .03$) and 0.62 ($SE = 0.07, p < .001$) increase in the number of distinct coding exercises solved, respectively.

6.1.3 Correlation between usage of PCEX and course performance. We also looked into the relationship between total usage of *PCEX* activities and the student's midterm or final grade, while controlling for the prior knowledge (i.e., pre-test score). Only the number of distinct coding exercises a student solved was a significant predictor of the midterm and final grade. Each successful attempt on coding exercises was associated with a 0.43 ($SE = 0.12, p < .001$) increase in the midterm and a 0.45 ($SE = 0.13, p < .01$) increase in the final grade.

6.2 Correlation between usage of PCEX and student's learning over time

6.2.1 Correlation analysis during the first and second half of the course . Research studying worked examples has consistently shown that the positive effect of worked examples is stronger in early stages of skill acquisition, when students typically have little or no domain knowledge, while gradually declining in later stages of skill acquisition as the learner develops more expertise [16, 28]. To investigate whether this relationship exists in our data, we split the data into halves, resulting in data from 55 students in the first half and 44 students in the second half of the course. We fitted regression models to predict the number of distinct coding exercises that student solved in each half using the *PCEX* activities completed and example line clicks. We also fitted regressions to predict the number of distinct challenges that student solved using the example line clicks. In all these regressions, we controlled for differences in pre-test scores.

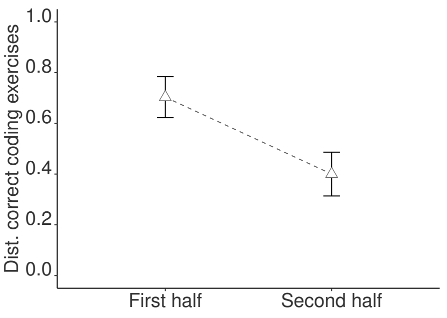
We plotted the estimated coefficients obtained from the regression analysis in Figure 3. Figure 3(a) shows the estimated coefficients for the *PCEX* activities completed. In both the first and second half of the course, *PCEX* activities completed was significant predictor of the distinct coding exercise solved. More specifically, in the first half, each *PCEX* activity completed was associated with a 0.7 ($SE = 0.1, p < .001$) increase in the number of distinct coding exercise solved. In the second half, each *PCEX* activity completed was associated with only a 0.4 ($SE = 0.1, p < .001$) increase in the number of distinct coding exercises solved (i.e., approximately half the early correlation).

The estimated coefficient for example line clicks was smaller than the coefficient for the *PCEX* activities completed (Figure 3(b)). It was significant only in the first half of the course and only for predicting the number of distinct challenges that a student solved. In the first half, each example line that was clicked increased the distinct correct attempts on challenges by 0.1 ($SE = 0.03, p = .01$). This coefficient was no longer statistically significant in the second half of the course, which suggests that individual line explanations accessible through line clicks are most important in the first half of the course when students are still in the early stages of learning. As students gain more knowledge in the domain, the knowledge added by each individual explanation becomes less essential.

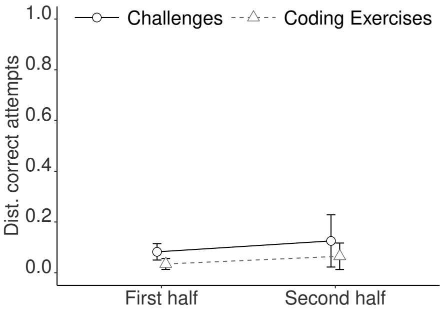
6.2.2 Correlation analysis of the regular and exam preparation usage. To further investigate how regularity of practice with the system influenced the learning results, we split the total practice of the students into *regular practice* during the semester and *exam preparation practice* (i.e., one week before the exam). Using spectral clustering, we grouped students based on the percentage of example lines clicked, the percentage of *PCEX* activities completed, and percentage of coding exercise solved[2]. We found three clusters that differed by the activity profile. The amount of practice within each cluster is shown in Figure 4(a) for regular practice and in Figure 4(b) for exam preparation practice.

Students in Cluster 1 had the highest amount of regular practice: On average, they completed 70% of the *PCEX* activities, solved 70% of the coding exercises and clicked on 10% of the example lines. Meanwhile, on average, the students in Cluster 2 clicked the same percentage of lines as students in Cluster 1 but completed 2.3 times fewer *PCEX* activities and solved 3.5 times fewer coding exercises. The students in Cluster 3 had the least amount of regular practice of all the clusters. On average, they clicked on 10% of the example lines, completed only 20% of the *PCEX* activities, and solved only 10% of the coding exercises. As seen in Figure 4(b), Cluster 2 was the only cluster that practiced with the system during the exam preparation week. Students in Cluster 2 had the same amount of practice during the exam preparation week as throughout the semester.

[2]We made sure that the variables used for clustering were not highly correlated (ρ was below 0.8 between each pair of variables).



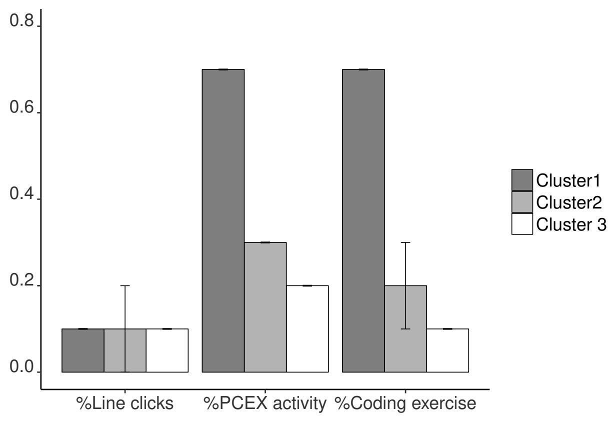
(a)



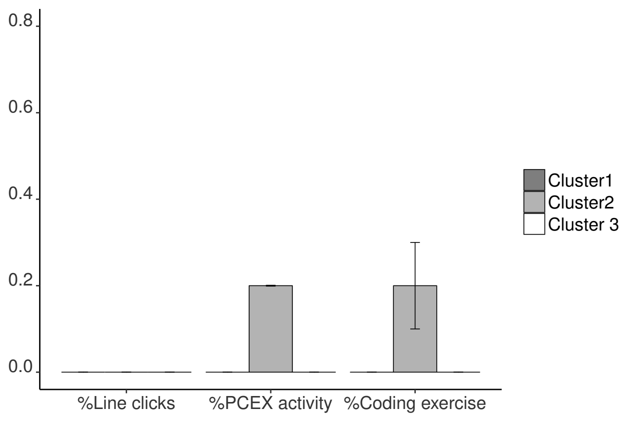
(b)

Figure 3: Regression estimates for (a) *PCEX* activities completed and (b) example line clicks on distinct problems that student solved during the first and second half of the course. Error bars show standard errors. In (b), the solid line represent the estimated coefficients for predicting the distinct challenges solved while the dashed line represents the estimated coefficients for predicting the distinct coding exercises solved.

Table 2 summarizes the learning results across different clusters. The learning results were analyzed using a one-way analysis of variance (ANOVA), followed by Tukey's post hoc comparisons. Overall, the correlation was significant for learning gain $F(2, 60) = 5.2, p < .01$ and midterm score $F(2, 60) = 4.9, p = .01$ but not on pre-test scores, ruling out the impact of initial differences between groups. Learning gain was significantly higher in Cluster 1, which included students with high regular practice (completing about 70% of the *PCEX* activities and coding exercises) compared to both Cluster 2 (moderate constant practice, $p = .01$) and Cluster 3 (low regular practice, $p = .02$). The regular practice in Custer 1 (high regular practice) is also associated with significantly higher midterm scores than Cluster 3 (low regular practice, $p = .01$), but only marginally higher than Cluster 2 (moderate constant practice, $p = .08$). These observations further suggest that students who worked with *PCEX* activities and coding exercises regularly obtained better learning results.



(a)



(b)

Figure 4: Percentage of practice for different clusters when system usage is split into (a) regular and (b) exam preparation. Usage is expressed as mean and standard error for the mean (error bars).

Table 2: Summary of learning results for clusters obtained after splitting practice into regular and exam preparation. Values are expressed as mean and standard error for the mean (in parentheses). The pretest score ranges from 0 to 29, learning gain ranges from 0 to 1, and midterm/exam score ranges from 0 to 100.

Cluster	Pre-test	Learning gain	Exam score	Midterm score
1 (N=18)	4.8 (1.5)	0.7 (0.05)	91.9 (2.2)	94.1 (1.3)
2 (N=19)	5.0 (1.4)	0.5 (0.06)	81.0 (4.6)	83.5 (3.6)
3 (N=25)	4.5 (1.0)	0.5 (0.05)	81.0 (3.9)	80.0 (3.6)

7 SURVEY ANALYSIS

We conducted a survey at the end of the semester to collect students' opinion of the PCEX activities. The survey consisted of two parts. In the first part, students responded about the amount of system use: *Yes-more than 10 times*, *Yes-between 5 and 10 times*, *Yes-less than 5 times*, and *No*. Those who chose one of the last two options were asked to provide their opinion on 6 follow-up items that focused on why the system was not used. Two of the items referred to *bad system experience*, two emphasized *no help needed*, and two addressed *other reasons*, in particular poor introduction of the system and lack of time to use the system.

The second part of the survey aimed to evaluate the PCEX activities, focusing on only students who used the system. Following the suggestion in [17] that identified key constructs required to evaluate a learning objective, we included three constructs: learning, quality, and engagement. Each construct had four items, two negatively and two positively worded. For the learning construct, items referred to student perception of how much they learned from using the PCEX activity. For the quality construct, items referred to the quality of the PCEX activity. Finally, for the engagement construct, items examined the level of student involvement in the PCEX activity. Note that in the survey, we referred to PCEX activities as examples-challenges because the practice system grouped PCEX activities under this name. Across all survey items, students were asked to respond using a 5-point Likert scale ranging from *Strongly Disagree* (1) to *Strongly Agree* (5).

In the first step of the survey analysis, we assessed the reliability of the survey items under each construct using Cronbach's α. We dropped two items from the engagement construct because item-construct correlations were lower than the recommended value, 0.30. Additionally, we checked whether the internal consistency could improve if any of the items within a construct were deleted. All items in the learning and engagement construct had acceptable internal consistency with the other items within that construct. The α was 0.8 for the learning construct and 0.6 for the engagement construct. After we discarded two items from the quality construct, the α improved from 0.7 to 0.9. No further item was discarded from the survey. At the end, all three constructs appeared to be sufficiently reliable to assess the value of PCEX activities, with α values exceeding the suggested minimum acceptable α coefficient of 0.50 [22].

Out of the 65 students who provided consent to use their data, 43% used the system more than 10 times, 37% used the system between 5 and 10 times, 14% used the system less than 5 times, and 6% did not use the system at all. The first plot in Figure 5 shows distribution of students answers to the 6 survey items that referred to the reasons for low/zero usage of the practice system. Overall, students mostly agreed that they did not use the system due to lack of time, preferring other resources and materials, and not feeling the need for additional help. Students disagreed with items that suggested other reasons for low/zero of the practice system, including the items that referred to bad system experience.

Distribution of students answers relating to the value of PCEX activities is shown in the second plot in Figure 5. The mean learning rating was 3.8 (SD = 0.8) which indicates that students perceived

PCEX activities to be helpful for learning. Notably, more than 70% of the students agreed with the items under the learning construct *(items 1-2, 4-5 in the y-axis)*. The mean quality rating was 3.7 (*SD* = 1), indicating that the students were also positive toward the quality of explanations and code in the PCEX activities *(item 3 and item 6 in the y-axis)*. The mean engagement rating was 3.2 (*SD* = .9), very close to the neutral part of the scale. While approximately 40% of the students agreed that they tried hard to understand the examples and challenges and did not skim-read them, a sizable fraction of the class disagreed with these statements *(last two items in the y-axis)*.

8 CONCLUSION AND FUTURE WORK

This paper introduced PCEX, an interactive tool to support learning from program construction examples. PCEX made each program example explorable and engaging: students can explore each example interactively and check their understanding by solving challenges that are similar to that example. To promote learning, the examples in PCEX are enriched by worked steps with subgoal labels and explanations. To assess the relationship between using this new educational technology and student's learning, the paper also reported results from an exploratory semester-long classroom study. In this study, students enrolled in a Java Programming class were encouraged to use a non-mandatory practice system, which included PCEX activities as well as automatically-assessed coding exercises. When analyzing the collected data, we observed that completing *PCEX* activities had a significant correlation with learning gain. Those students who completed more *PCEX* activities learned more than those who completed fewer (or no) *PCEX* activities. We also found that work with *PCEX* activities had a positive correlation with student's performance in coding exercises even when prior knowledge (as measured by the pre-test) was controlled.

Another interesting observation was that the correlation of work with *PCEX* activities and student's learning was stronger in the first half of the course than the second half. This finding is consistent with past studies that showed that the worked example effect is stronger in the early stages of learning and declines as a student's knowledge grows [16, 28].

We also found that regular practice with *PCEX* activities is associated with better learning outcomes. Students who used *PCEX* activities regularly during the course and, on average, completed 70% of the *PCEX* activities and coding exercises, achieved higher learning gain and midterm score than students in the group that used the *PCEX* activities and coding exercises less regularly but worked more during the exam preparation time.

Finally, the survey results suggest that students found the *PCEX* activities to be of high quality and helpful for learning programming. At the same time, students' self-reported level of engagement with *PCEX* activities was lower than other aspects of their feedback. This indicates a need for follow-up interviews to uncover possible reasons for lower engagement and discuss options to improve the engagement side of *PCEX*.

Bringing together two sources of information (logs and survey), we can also observe that both approaches to augment traditional examples, line-level explanations and challenges, were valuable for the students. As shown in Section 6.1, every explored line explanation and every attempted challenge can be associated with an improvement in student's performance. Students' feedback on survey items that separately assessed the educational value of explanations and challenges was also positive in both cases. At the same time, the data hints that among these two types of augmentation, the challenges are more valuable educationally and more appealing to the students. As Section 6.1 shows, the positive correlation of one explored line was more than 10 times lower (0.05 vs 0.62) than the positive correlation of one challenge (which typically expected students to move 2 – 4 lines of code). Students' feedback about

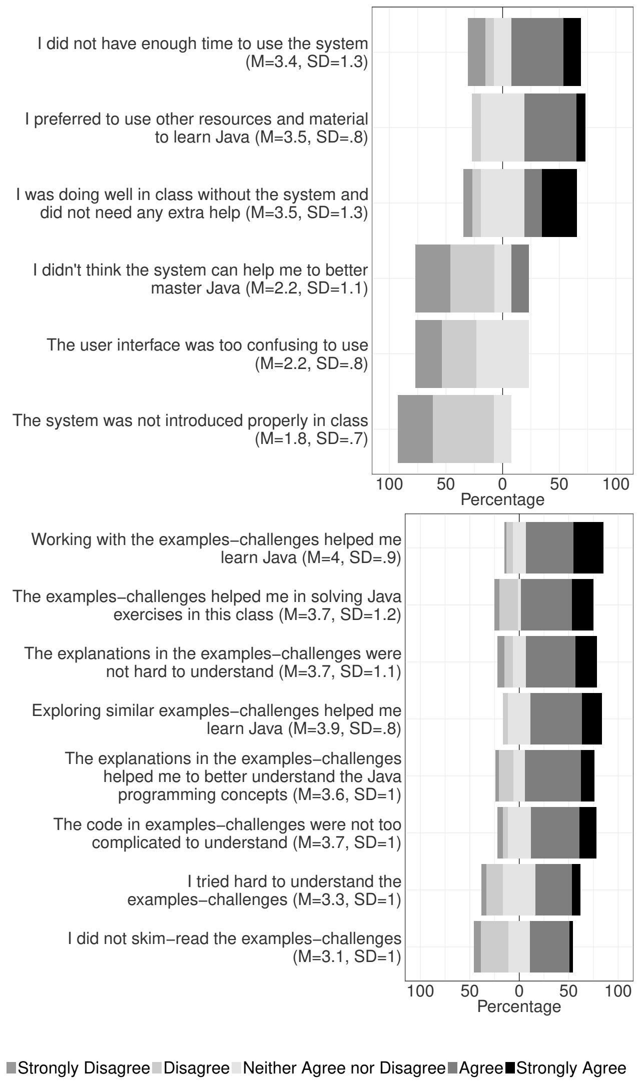


Figure 5: The distribution of answers for the survey items. The percentage of respondents who agree/disagree with each item are shown to the right/left of the zero line. The percentage of respondents who neither agree nor disagree are split down the middle and are shown in a neutral color. The items in the y-axis are ordered based on the percentage of agreements, with the uppermost/lowermost item having the most/least agreement.

the value of explanations was also slightly less positive than their feedback about challenges (3.6 vs 3.9). The same trend can also be observed in the usage statistics reported in Section 5: the students completed 51% of the PCEX activities, but explored only 11% of the example lines.

An important goal of this study was to evaluate long-term impact of our technology in the target context. While the format of a semester-long study in a real college course was the best match to our goal, it also made it challenging, practically and ethically, to form a control group with only partial access to the system (i.e., no examples or no challenges). As a result, our study has at least two important limitations. First, our results might be subject to a self-selection bias. Since the amount of PCEX usage was determined by the students themselves, we cannot determine whether PCEX helped students with getting a better grade, or if, instead, students with a better grade were more motivated to succeed and, as a result, allocated more time to work with the non-mandatory practice system. Second, our design combined explorability and challenges in a single condition, making it impossible to assess the impact of each of these features.

In future work, we plan to conduct a randomized controlled study with students practicing programming with/without PCEX as well as with/without explorability and challenges. We expect to confirm that the combined effect of explorability and challenges would improve learning from examples and hope to assess their comparative impact more reliably. We also plan to assess the impact of PCEX examples on student learning in other domains, such as Python programming. Finally, we plan to develop an authoring tool to make PCEX available to broader community.

ACKNOWLEDGMENTS

This work was presented as part of doctoral dissertation of the first author at the University of Pittsburgh, entitled *"Program Construction Examples in Computer Science Education: From Static Text to Adaptive and Engaging Learning Technology"* [13]. The dissertation is available at http://d-scholarship.pitt.edu/35280/.

REFERENCES

[1] Robert K Atkinson, Sharon J Derry, Alexander Renkl, and Donald Wortham. 2000. Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research* 70, 2 (2000), 181–214.

[2] Peter Brusilovsky and Colin Higgins. 2005. Preface to the special issue on automated assessment of programming assignments. *ACM Journal on Educational Resources in Computing* 5, 3 (2005), Article No. 1.

[3] Kevin Buffardi and Stephen H. Edwards. 2014. Introducing CodeWorkout: An Adaptive and Social Learning Environment (Abstract Only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 724–724. DOI:http://dx.doi.org/10.1145/2538862.2544317

[4] Jennifer Campbell, Diane Horton, and Michelle Craig. 2016. Factors for Success in Online CS1. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 320–325. DOI:http://dx.doi.org/10.1145/2899415.2899457

[5] Jennifer Campbell, Diane Horton, Michelle Craig, and Paul Gries. 2014. Evaluating an Inverted CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 307–312. DOI:http://dx.doi.org/10.1145/2538862.2538943

[6] Richard Catrambone. 1998. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General* 127, 4 (1998), 355.

[7] Steve Cooper and Mehran Sahami. 2013. Reflections on Stanford's MOOCs. *Commun. ACM* 56, 2 (Feb. 2013), 28–30.

[8] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 471–476. DOI:http://dx.doi.org/10.1145/1953163.1953299

[9] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.* 40, 3 (June 2008), 328–328. DOI:http://dx.doi.org/10.1145/1597849.1384371

[10] Barbara Ericson, Steven Moore, Briana Morrison, and Mark Guzdial. 2015. Usability and Usage of Interactive Features in an Online Ebook for CS Teachers. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15)*. ACM, New York, NY, USA, 111–120. DOI:http://dx.doi.org/10.1145/2818314.2818335

[11] Julio Guerra, Christian D Schunn, Susan Bull, Jordan Barria-Pineda, and Peter Brusilovsky. 2018. Navigation support in complex open learner models: assessing visual design alternatives. *New Review of Hypermedia and Multimedia* (2018), 1–29.

[12] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-based Program Visualization for CS Education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 579–584. DOI:http://dx.doi.org/10.1145/2445196.2445368

[13] Roya Hosseini. 2018. *Program Construction Examples in Computer Science Education: From Static Text to Adaptive and Engaging Learning Technology.* Ph.D. Dissertation. University of Pittsburgh.

[14] Roya Hosseini, Teemu Sirkiä, Julio Guerra, Peter Brusilovsky, and Lauri Malmi. 2016. Animated examples as practice content in a java programming course. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* ACM, 540–545.

[15] David Hovemeyer, Matthew Hertz, Paul Denny, Jaime Spacco, Andrei Papancea, John Stamper, and Kelly Rivers. 2013. CloudCoder: building a community for creating, assigning, evaluating and sharing programming exercises. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education.* ACM, 742–742.

[16] Slava Kalyuga, Paul Ayres, Paul Chandler, and John Sweller. 2003. The expertise reversal effect. *Educational Psychologist* 38, 1 (2003), 23–31.

[17] Robin H Kay and Liesel Knaack. 2009. Assessing learning, quality and engagement in learning objects: the Learning Object Evaluation Scale for Students (LOES-S). *Educational Technology Research and Development* 57, 2 (2009), 147–168.

[18] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, and others. 2004. A multi-national study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 119–150.

[19] Tomasz D Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. 2014. Mastery grids: An open source social educational progress visualization. In *European Conference on Technology Enhanced Learning*. Springer, 235–248.

[20] Briana B Morrison, Lauren E Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals help students solve Parsons problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* ACM, 42–47.

[21] Thomas Naps, Stephen Cooper, Boris Koldehofe, Charles Leska, Guido Rößling, Wanda Dann, Ari Korhonen, Lauri Malmi, Jarmo Rantakokko, Rockford J Ross, and others. 2003. Evaluating the educational impact of visualization. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 124–136.

[22] Jum C Nunnally and Ira H Bernstein. 1978. Psychometric theory. (1978).

[23] Nick Parlante. 2017. codingbat.com. (2017). http://codingbat.com/about.html Accessed on Jan 21, 2018.

[24] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.* Australian Computer Society, Inc., 157–163.

[25] Rmi Sharrock, Ella Hamonic, Mathias Hiron, and Sebastien Carlier. 2017. CODECAST: An Innovative Technology to Facilitate Teaching and Learning Computer Programming in a C Language Online Course. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale.* ACM, 147–148.

[26] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 15.

[27] Juha Sorva and Teemu Sirkiä. 2010. UUhistle: A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. ACM, New York, NY, USA, 49–54. DOI:http://dx.doi.org/10.1145/1930464.1930471

[28] John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. 1998. Cognitive architecture and instructional design. *Educational Psychology Review* 10, 3 (1998), 251–296.

[29] Brad Vander Zanden, David Anderson, Curtis Taylor, Will Davis, and Michael W. Berry. 2012. Codeassessor: An Interactive, Web-based Tool for Introductory Programming. *J. Comput. Sci. Coll.* 28, 2 (Dec. 2012), 73–80. http://dl.acm.org/citation.cfm?id=2382887.2382900

[30] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. 2013. Facilitating Code-writing in PI Classes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 585–590. DOI:http://dx.doi.org/10.1145/2445196.2445369