International Journal of Artificial Intelligence in Education (2020) 30:299–336
https://doi.org/10.1007/s40593-020-00197-0

ARTICLE

Improving Engagement in Program Construction Examples for Learning Python Programming

Roya Hosseini[1] · Kamil Akhuseyinoglu[1] · Peter Brusilovsky[1] · Lauri Malmi[2] · Kerttu Pollari-Malmi[2] · Christian Schunn[1] · Teemu Sirkiä[2]

Published online: 17 June 2020

Abstract

This research is focused on how to support students' acquisition of program construction skills through worked examples. Although examples have been consistently proven to be valuable for student's learning, the learning technology for computer science education lacks program construction examples with interactive elements that could engage students. The goal of this work is to investigate the value of the "engaging" features in programming examples. We introduce PCEX, an online tool developed to present program construction examples in an engaging fashion. We also present the results of a controlled study with a between-subject design that was conducted in a large introductory Python programming class to compare PCEX with non-interactive worked examples focused on program construction. The results of our study show the positive impact of interactive program construction examples on student's engagement, problem-solving performance, and learning.

Keywords Introductory programming education · CS1 · Python · Program construction · Worked examples · Classroom study

Introduction

Programming 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 and to highlight fundamental coding patterns. Programming textbooks also place a heavy emphasis on examples, with a large proportion of textbook space being devoted to program examples and associated comments. Moreover, the code of all presented examples is typically provided on an accompanying CD or website to encourage students to explore, run, and modify the examples. Finally,

✉ Roya Hosseini
roh38@pitt.edu

Extended author information available on the last page of the article.

recent studies have also shown that, on many occasions, students prefer seeing examples to receiving hints on how to build the code and fix their programs (Rivers 2017).

To increase the overall pedagogical value of code examples, textbooks and e-learning tools frequently augment the code with additional information that focuses on building either *program comprehension* or *program construction* skills. Program comprehension is related to reading, comprehending, and tracing the code, while program construction is related to writing or building the code. However, e-learning tools tend to support these two types of programming skills quite unevenly. While there is an active stream of work that has focused on augmenting code examples to support program comprehension (i.e., dynamic code animations and interactive program visualizations (Sorva et al. 2013)), there are comparatively few e-learning tools focused on augmenting code examples with the goal of supporting program construction skills (i.e., a step-by-step demonstration on how to solve a specific programming problem (Sharrock et al. 2017)).

The most popular approach to present program construction examples is text-based *worked examples*. A typical worked example, as presented in the majority of printed and online textbooks, focuses on a meaningful programming problem to solve, presents code of a sample solution, and augments this code with a textual explanation of the code design and implementation process shown in steps with various granularity. While these explanations include all necessary information, it is presented in a static and non-engaging form. Some recent work has attempted to address the static textual nature of program construction examples by offering tools to construct interactive explanations (Brusilovsky and Yudelson 2008) and codecasts (Sharrock et al. 2017); yet the technology for presenting program construction examples is lagging behind the state-of-the-art in presented program comprehension examples that actively engage students in the work with examples (Hundhausen et al. 2002; Naps 2005). In contrast, the research on program-construction problems has recently explored a number of interactive approaches to engage students in program construction activities, such as the modification of code elements or adding missing parts in the code using, for instance, the Activecode elements (Miller and Ranum 2012; Ericson et al. 2015) or Parsons problems (puzzles) approach (Parsons and Haden 2006; Ihantola and Karavirta 2011). Such activities can range from smaller-scale program completion to constructing fully working programs. Yet, unlike worked examples, program-construction problems offer no explanations and have focused more on knowledge assessment than knowledge acquisition.

In our work, we attempted to combine the explanatory power of traditional worked examples with the elements of interactivity and engagement offered by interactive examples and programming problems. We believe that our work bridges the gap between these two categories of learning activities and forms a continuum from static textbook examples into actual full-coding exercises. We introduce PCEX, an online learning tool that focuses on presenting program construction examples to students. PCEX offers examples in an interactive, engaging form in order to increase students' motivation to work with examples and improve their overall learning.

We formulated the following research questions in order to build a better understanding of the impact of PCEX on learning programming and to compare the impact of this interactive style to that of non-interactive examples. In this study, the non-interactive examples we presented were textbook-style worked examples that focused on program construction skills. By textbook-style worked examples, we mean the static worked examples typically presented in textbooks, which often lack interactivity and a challenge component. Specific hypotheses related to the research questions are presented in "Hypotheses":

[RQ1.] Will working with PCEX examples engage students more than working with textbook-style worked examples? **[Hypothesis 1]**

[RQ2.] Will working with PCEX examples lead to better performance in solving program construction problems than working with textbook-style worked examples? **[Hypothesis 2]**

[RQ3.] Will working with PCEX examples lead to better learning outcomes than working with textbook-style worked examples? **[Hypothesis 3]**

In this article, we address these research questions by presenting results from a semester-long classroom evaluation that we conducted in a large introductory programming class to compare PCEX with textbook-style worked examples that focused on program construction. Our results demonstrated the benefits of PCEX examples over textbook-style worked examples on student's engagement, performance, and overall learning outcomes.

The remainder of this article is organized as follows. After reviewing the related work in the next section, we describe our proposed tool, PCEX, in "Engaging Program Construction Examples". Section "The Study" describes our study design, while "Results" presents the results of the study. Section "Summary and Discussion" concludes with a summary and discussion of this work and direction for future research.

Background and Related Work

Worked Examples in Problem-Solving

Over the last 30 years, worked examples, also referred to as worked-out examples (Atkinson et al. 2000), have gradually emerged as an important instructional approach that is supported by learning technology. A sizable body of research on instructional practices that support the use of worked examples for acquiring cognitive skills has been accumulated in various domains, such as mathematics and physics (Atkinson et al. 2000; Chi et al. 1989; Sweller and Cooper 1985). Worked examples are comprised of the presentation of a problem, the solution steps, and the final solution. Students use these examples as models of how to solve certain types of problems. Research on studying worked examples has consistently shown that in the early stages of skill acquisition, when students typically have little or no domain knowledge, instruction that relies more heavily on studying worked examples is more effective for learning than the traditional approach of

being focused on only problem-solving. It has been shown that early example-based instruction leads to better learning outcomes, which are reached in less time and with less effort. This is usually referred to as the "worked example effect" (Sweller et al. 1998).

The examples appear to be most valuable when presentation of the examples is combined with problem-solving. Specifically, past studies found that the pairing of worked examples with practice problems is more effective than providing learners with only practice problems (Sweller and Cooper 1985) or only examples (Trafton and Reiser 1993). During the later stages of skill acquisition, however, the positive effect of worked examples gradually declines. In fact, example-based learning is inferior to problem-solving when learners have gained a reasonable degree of domain knowledge (Kalyuga et al. 2000, 2001, 2003). While learning from examples is superior to problem-solving for learners with little domain knowledge, this advantage disappears over time as the learners develop more content expertise. This phenomenon is referred to as the "expertise reversal effect" (Kalyuga et al. 2003). On one hand, this research points out the importance of examples for less prepared students, while on the other hand, it stresses the importance of carefully adapting to the current level of the student's knowledge by decreasing the number of worked examples as the student gains expertise. A successful approach to implement this is using "faded worked examples" where the supportive guidance for students is gradually reduced as the student moves towards fully independent problem-solving (Renkl and Atkinson 2007). A further improvement is *adaptive fading*, where the level of fading and/or the choice between a regular worked example, a faded example, or a problem is determined based on the students' understanding of the examples (Salden et al. 2010a) or their performance (Najar et al. 2016). More recent research explored an adaptive alternation of worked examples, problem-solving, and *erroneous examples* (Chen et al. 2019a, b), a promising alternative to the faded examples (McLaren et al. 2012).

According to Nokes-Malach et al. (2013), worked examples are hypothesized to be effective for several reasons. First, they provide constraints to the solution space; i.e., they highlight the correct solution path so students do not need to waste time on incorrect or unfruitful searches. As a result, novices obtain the information needed to gain generalized knowledge more quickly (Salden et al. 2010b). Second, they reduce irrelevant cognitive load by highlighting the important elements of the problem and solution for the learner to focus on, encode, and reason about (Paas and Van Merriënboer 1994; Ward and Sweller 1990). Third, they encourage constructive cognitive processes such as self-explanation, in which the learner explains to herself the underlying conceptual logic and justification behind each step (Catrambone 1998; Renkl 1997). This may be particularly important for helping students link the features of an example to abstract domain concepts or underlying principles.

It is worth noting that while the example-based learning approach often fosters learning for students with low levels of prior knowledge, this outcome is not guaranteed (Atkinson and Renkl 2007). The positive impact of examples for the student's learning can only be observed when (a) examples are designed effectively, and (b) students study the examples thoroughly.

Worked Examples in Programming

While worked examples are generally less explored in the domain of programming, there is a reasonable body of research that has guided our work. The earliest successful research showing the value of examples for learning programming dates back to the early 1980s, when Pirolli and Anderson (1985) reported that examples are helpful for guiding students to solutions for novel and difficult problems. Since then, a considerable amount of research has been devoted to the development of example-based learning environments to support students in learning programming in various programming languages, such as LISP (Getao 1990; Lieberman 1987; Weber 1996; Weber and Brusilovsky 2001; Weber and Mollenberg 1994), Prolog (Brna 1998), C/Java (Loboda and Brusilovsky 2010; Esteves and Mendes 2003; Sirkiä 2013; Brusilovsky and Yudelson 2008), Javascript (Davidovic et al. 2003), SQL (Najar et al. 2016; Chen et al. 2019a, b), and mini-languages (Brusilovsky 1994; Zhi et al. 2019).

We classify program examples that have been used in teaching and learning to program 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 programming 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 the construction of a program that achieves various meaningful purposes (e.g., summing an array). This distinction might not be clear-cut for code-only examples, since the same code could be used for both purposes. However, attempts to augment examples with learning technologies to increase their instructional value (i.e., adding code animation or explanations) usually focus on one of these goals.

Program behavior examples have been extensively studied. While textbooks and tutorials still explain program behavior by using textual comments attached to lines of program code, a more advanced method for this purpose — *program visualization*, which visually illustrates the runtime behavior of computer programs — is becoming increasingly 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 (Sorva et al. 2013). Despite their visual and dynamic nature, the majority of tools for presenting animated examples, especially the first generations, could be considered non-engaging, in that they limit the student's role to passively watching the animation (e.g., Miyadera et al. (2007), Sajaniemi and Kuittinen (2003), and Sirkiä (2013)). Following several studies that have demonstrated the low effectiveness of "passively-watched" animated examples, several researchers have experimented with interactive animations that are explorable and challenging, for example, allowing the student to change input data (Lawrence 1993), asking students to predict the result of a specific step (Byrne et al. 1999), or asking strategic questions about the visualization (Hansen et al. 2000; Naps et al. 2000). A meta-study or first-generation research on program visualization (Hundhausen et al. 2002) helped to stress the value and the importance of interactive engagement in the context of animated examples. With the help of this analysis, an international working group attempted to classify several known types of interactive engagement (such as

viewing, *responding*, *changing*, or *constructing*) by its cognitive level or depth associating deeper engagements with higher levels of taxonomy of educational objectives (Bloom 1956). Since that time, a considerable volume of empirical evidence confirmed that that more active and deeper engagement of students into working with visualized program examples improved their learning (Byrne et al. 1999; Hundhausen et al. 2002; Myller 2006; Lawrence 1993; Naps 2005; Sears and Wolfe 1995) and had a positive influence on their problem-solving abilities in that domain (Evans and Gibbons 2007).

Advanced technologies for presenting program construction examples are much less developed. In contrast to interactive and engaging worked examples in math and physics, for many years, the dominant approach for presenting worked code examples was simply text with comments (Linn and Clancey 1992; Davidovic et al. 2003; Morrison et al. 2016). More recently, this technology has been enhanced by adding audio narrations to explain the code (Ericson et al. 2015) or by showing video fragments of code screencasts with the instructor's narration being heard while watching slides or an editor window (Sharrock et al. 2017). Both approaches, however, can still be considered as a passive presentation that does not allow for exploration and engagement. An attempt to add interactivity and exploration to worked program construction examples in the form of commented code was made in the WebEx system, which allows students to interactively explore line-by-line comments for program examples via a web-based interface (Brusilovsky et al. 2009). More recently, (Khandwala and Guo 2018; Park et al. 2018) attempted to make the screencast-based examples more engaging by allowing inline code editing or embedding programming exercises into the videos. However, with the notable exception of building faded examples for SQL (Najar et al. 2016), existing approaches for presenting program construction examples lack the engagement power of modern technologies for presenting program behavior examples. In this work, we attempt to use research findings in the area of program behavior examples to produce interactive program construction examples, called PCEX, that better engage students and improve their learning.

Block-Based Programming and Parsons Problems

Block-based programming environments adopt a drag-and-drop interface that allows users to construct and execute computer programs by composing atomic blocks of code together. Common examples of block-based programming environments are Scratch (Resnick et al. 2009); Snap, a web-based environment based on Scratch (Harvey and Mönig 2010); and Alice. Scratch programs allow users to build the program in small chunks, execute any piece of program code individually, and see its effects immediately. Alice 2 (Cooper et al. 2003) and its successor Alice 3 (Dann et al. 2012) apply an event-based object-oriented model. In Alice, users can program within a 3D environment, add objects to the scene, manipulate the objects' attributes, and call the object's methods.

Parsons problems are another related research to block-based programming. Parsons and Haden (2006) originally created Parsons problems as an easy way for novices to solve programming assignments without having to type code or think about the exact syntax of the programming language. The idea is clever: a small

number of code fragments are presented in a random order and the novice is asked to construct the described function or a small program by placing the fragments in the correct order. Each fragment may contain one or more lines of code and all of the fragments may not be required in the solution. In Ihantola and Karavirta (2011), the authors introduced a new family of Parsons problems inspired by the Python programming language. They proposed a two-dimensional variant of Parsons problems where lines of code are not only sorted, but also placed on a two-dimensional surface. The vertical dimension is used for ordering the code, as in traditional Parsons problems. The horizontal dimension is used to define code blocks, based on indentation. In another attempt, Ericson et al. (2017) proposed a 2D Parsons problem with paired distractors where each distractor was shown paired with the matching correct code block so that the learner only had to choose the distractor or the correct code. More recently, Fabic et al. (2019) implemented and explored a version of Parsons problem with internal re-shuffling for mobile phones.

Engaging Program Construction Examples

This section introduces PCEX, an online tool developed to present program construction examples in an engaging fashion. First, it defines exactly what the student's engagement is, what are its different constructs, and which of the engagement constructs are used in the PCEX examples. Then, it presents the interface and discusses which interactivity elements are used in the PCEX examples to engage students.

Targeted Aspects of Engagement

Engagement is a construct that has many different definitions in education, ranging from activity completion to particular cognitive and affective forms of activity completion. Therefore, we need to define our conception of student's engagement to ground our approach to creating examples that support engagement. We define engagement as the extent of a student's active involvement in a learning activity (Christenson et al. 2012). It is often considered to be a multi-dimensional construct of possibly four distinct, yet inter-correlated and mutually supportive aspects: *behavioral engagement*, *emotional engagement*, *cognitive engagement*, and *agentic engagement* (Christenson et al. 2012; Reeve 2013; Reeve and Tseng 2011). Each of these dimensions offers different ways to measure student's engagement; although, as mentioned above, these measures are not fully independent.

Behavioral engagement refers to how effortfully the student is involved in the learning activity, in terms of attention, effort, and persistence (Christenson et al. 2012; Skinner et al. 2009). Higher behavioral engagement is associated with larger amount of work that the students are willing to perform, higher levels of completion and retention, and higher persistence in problem-solving. These aspects could usually be measured by processing students' logs. Emotional engagement refers to the presence of positive emotions during task involvement, such as interest, and to the absence of negative emotions, such as anxiety (Skinner et al. 2009). Higher levels of emotional engagement could be recognized by higher levels of student's satisfaction,

bringing a more positive attitude to the learning process, and a willingness to recommend a novel learning tool to peers. These aspects could be measured through both surveys and questionnaires. Cognitive engagement refers to how strategically the student attempts to learn in terms of using sophisticated rather than superficial learning strategies, such as elaboration rather than memorization (Walker et al. 2006). Cognitive engagement is defined to a considerable extent by the design of learning activities. In particular, the taxonomy of educational objectives (Bloom 1956) was designed to help instructors in creating educational activities with higher levels of cognitive engagement (such as activities based on responding, changing, or constructing in the case of animated program examples). Agentic engagement is a fourth and newly proposed aspect of student's engagement that refers to the extent of the student's constructive contribution to the flow of the instruction in terms of asking questions, expressing preferences, and letting the teacher know what the student wants and needs (Reeve 2013).

The work on engaging examples presented in this paper focuses predominantly on behavioral and cognitive engagement, which are the aspects that are most consistently linked with learning outcomes (Bathgate and Schunn 2017). Our design focuses on cognitive engagement: we sought to design examples that would more actively involve students in working through the full examples and encourage them to think more deeply. In accordance with past research on engagement (Skinner et al. 2009), we expect that higher cognitive engagement will be associated with higher behavior engagement and better learning outcomes. To better assess the increased behavior engagement, we evaluate our technology in a *voluntary practice* context. In our study we separate *compulsory* learning content, which the students are required to work with to receive their course grade and *voluntary* learning content, which the students are encouraged, but not required to work with. The presence of this voluntary content allows us to more reliably assess the content-influenced behavior engagement (working with compulsory content is not a reliable measure since many students are working only to "get the points"). In our study, this voluntary content allows students to explore interactive examples, as well as work with additional challenges where they can apply their knowledge through related problem-solving tasks. In this context, behavioral engagement could be measured by counting how many tasks or steps they carry out with the voluntary content, as well as how much time they spend working with the content. The details of measures are explained in "Metrics". To more reliably measure student's behavior engagement with voluntary content, we performed our experiment as a semester-long study in a real (and large) class. This design choice limited our ability to measure overall levels of emotional and agentic engagement, since neither long surveys nor active student-teacher communication were feasible in this context. While we expect that our technology could also lead to higher emotional and agentic engagement, we will need different kinds of experiments to confirm these expectations.

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" (Fig. 1a) and worked program steps (Fig. 1b). The goal states the function that the example program performs. The worked steps begin with a subgoal label (Fig. 1c) and are represented in the form of the 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's performance by leading students to group a set of steps and encouraging them to self-explain the reason for clustering those steps (Catrambone 1998). The example is enriched with instructional explanations that are indicated by question mark icons next to either all or a subset of example lines (Fig. 1d). Once a student clicks on a question mark, an explanation is shown on the right side (Fig. 1e). The student can request additional details for the selected line by clicking on the "Additional Details" button (Fig. 1g) or can navigate to the previous or next line to read an explanation (Fig. 1f).

In addition to being *explorable*, PCEX examples *challenge* students by engaging them with a problem-solving activity. When a student clicks on the "Challenge me" button (Fig. 1h), an interactive challenge activity is presented to the student, as shown in Fig. 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 similar to the original example, in both the goal to achieve and the code. A challenge has both a problem statement (Fig. 2i) and code. However, the code has no explanation and is not complete — one or more of the code lines are missing. The student's goal is to complete the code by dragging and dropping lines from the set of options (Fig. 2j) into each of the missing fields. This drag-and-drop interaction

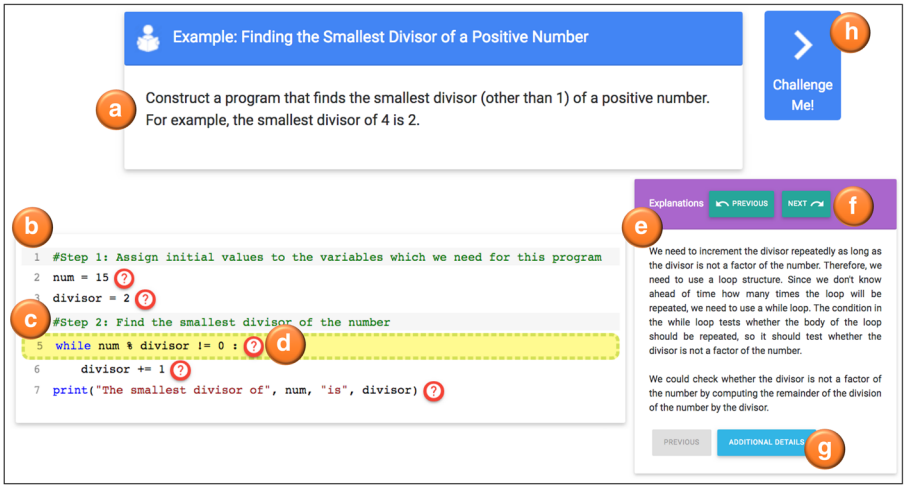

Fig. 1 A Python programming worked example in the PCEX activity. The example includes the goal (**a**), interactive worked code (**b**), the subgoal label presented as a comment (**c**), the link to instructional explanations (question mark symbols) (**d**), explanations (**e**), a navigation link to the explanation for the previous/next line (**f**), additional details for the highlighted line (**g**), and a challenge navigation link (**h**)

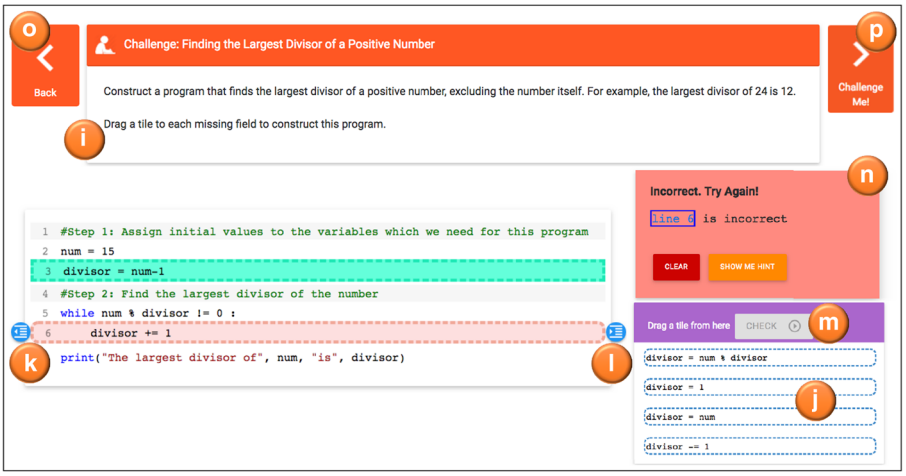

Fig. 2 A Python programming challenge in the PCEX activity that follows the worked example shown in Fig. 1. The challenge includes the goal (**i**), has one or more missing lines, and asks the student to drag and drop a line from the given options (**j**) to each missing line to construct the program. The student can decrease/increase the indentation of the line by using the indentation buttons (**k**)/(**l**). A student can request feedback by pressing the "Check" button (M), and the feedback message is shown in part (**n**). The student can go back to the example by pressing the "Back" button (**o**). If there are more challenges available, the student can go to the next challenge by pressing the "Challenge Me" button (**p**). This button is shown only when the current challenge is solved or after the student checks the solution after the third incorrect attempt

approach is similar to Parsons problems (Parsons and Haden 2006). The student can decrease/increase the indentation of the line by using the indentation buttons (Fig. 2k and l), and the student can check whether the challenge is solved correctly by clicking on the "Check" button (Fig. 2m). 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 (Fig. 2n). If the student cannot solve the challenge within three attempts, he or she can request the solution.

At any moment, the student can navigate to the core explained example by pressing the "Back" button (Fig. 2o). Also, if an example has several challenges, the student may navigate between them by pressing the "Challenge Me" button (Fig. 2p). 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.

Implementation Details

In this work, we use the term *PCEX activity* to refer to the PCEX worked example and its associated challenges. PCEX activities are created by annotating the example and challenge code with a set of predefined tags (see Fig. 3). These tags are used to define the goal or the problem statement, the instructional explanations, the lines to be faded, the set of line options, and the user input to the program. We used *docstrings* for Python or *doc comments* for Java to add these annotations to the program code.

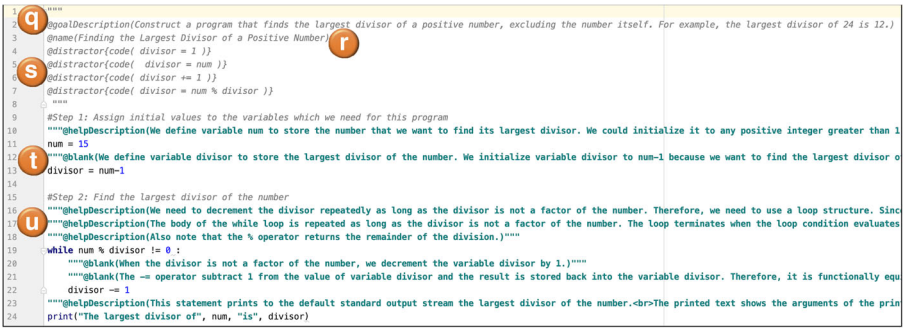

Fig. 3 An annotated Python programming code for the PCEX challenge shown in Fig. 2. The annotated code includes the problem statement (@goalDescription, q), the title of the challenge (@name, r), the set of option lines (@distractor, s), the lines to be faded (@blank, t), and the instructional descriptions (@helpDescription, u). The annotation tags are inserted using Python docstrings

The annotated code is parsed to generate a corresponding JSON file for each of the PCEX activities. The JSON file is then used by a single-page JavaScript application to support interactive work with the examples and challenges as shown in Figs. 1 and 2. The JavaScript application only needs the parsed JSON file to operate functionally, and it communicates back to the student modeling server (Yudelson et al. 2007) to log students' interactions. The current version of the PCEX supports any executable code in both Java and Python.

PCEX employs an output-based evaluation of students' solutions. As a result, the program code of a challenge is required to produce an output. The JSON file generated by the parsing process contains the correct program output (i.e. ideal solution) and the program outputs of all possible students' solutions. This is achieved by first completing the code with all permutations of the line options (Fig. 2j) and then by running each completed code. This way, we can identify the cases where the student's solution is correct, even if the solution consists of a different option order than the ideal solution. Moreover, this approach helps us to promptly provide feedback to the student, as opposed to waiting for the program execution to finish each time the student clicks on the "Check" button (Fig. 2m).

The Study

To evaluate the impact on student's engagement and learning from adding explorability and challenges to program construction examples, we conducted a classroom study followed by a survey to collect students' feedback on the overall usefulness of the new program construction examples. Using a classroom study format was essential for our evaluation, since assessing engagement issues requires a realistic context. While the less controlled nature of a classroom study makes it harder to assess the overall impact of learning technology on learning gain, this type of study enables us to more reliably measure the engagement and motivation side of student's work. In this section, we present details of the study, beginning by explaining the design of the

study, then presenting the information about participants and the study procedure. We conclude this section with information about the objective and subjective measures that were collected in the study and are later used in the data analysis.

Hypotheses

This study tests the following hypotheses regarding the benefits of PCEX compared to textbook-style, static, worked examples:

*H*1. Work with PCEX examples would engage students more than the work with textbook-style worked examples;
*H*2. Work with PCEX examples would lead to better performance in solving program construction problems than the work with textbook-style worked examples; and
*H*3. Work with PCEX examples would lead to better learning outcomes than the work with textbook-style worked examples.

Study Design

To test our hypotheses, we designed a classroom study with two groups. In both groups, students could practice with voluntary examples and problems that were accessible through an online practice system. In order to make the groups equivalent in terms of the quantity and quality of learning material, both groups also received the same problems, program examples, and explanations. The difference between the groups was only in the way that examples were presented to the students. In the Experimental group, students could practice with PCEX activities that presented the sequence of similar examples with the first example fully worked (Fig. 1) and the rest of the examples presented as tasks that challenged students (Fig. 2). In the Control group, in contrast, students could practice with textbook-style example activities (see "Control Condition Interface: Textbook-Style Worked Examples") that presented similar worked examples in the same order, all in the same form as shown in Fig. 4.

Control Condition Interface: Textbook-Style Worked Examples

The Control condition interface presents the examples in a traditional textbook-style form using a simple technology that shows examples statically. Figure 4 illustrates a worked example that was presented in this static interface. Each example includes a "goal" (Fig. 4a) and worked program steps (Fig. 4b). To make the presentation of the worked examples similar to those in programming textbooks, code segments and explanations are interleaved by default. The interface also enables the student to have a view of the code, similar to how to the code is presented in an integrated development environment (IDE). The student can click on the "Hide Explanations" button (Fig. 4c) to switch to the mode that presents the code with no explanations. Figure 5 illustrates the code-only mode, with explanations hidden. The student can switch back to the mode where the explanations are shown by clicking on the "Show

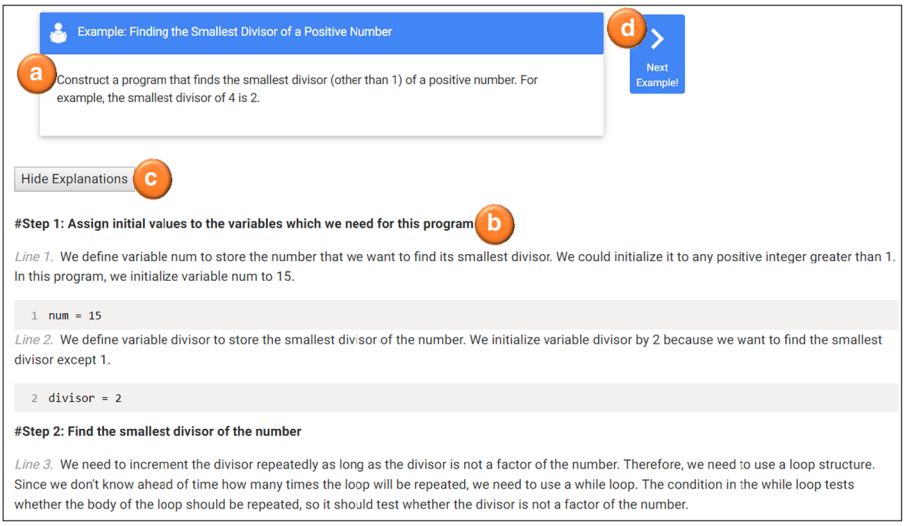

Fig. 4 The default mode of a worked example in the Control interface that presents static worked examples for program construction skills. The example is presented in a textbook-style form and includes the goal (**a**), worked code with subgoal labels and explanations (**b**), the link to hide explanations (**c**), and a navigation link to the next similar example (**d**). When appropriate, a previous example arrow will appear on the left side

Explanations" link (Fig. 5e). The student can go to the next similar example by clicking on the "Next Example" link (Fig. 4d). The next example will be presented in the same style as the previous example. The student can navigate between similar examples by using the navigation links.

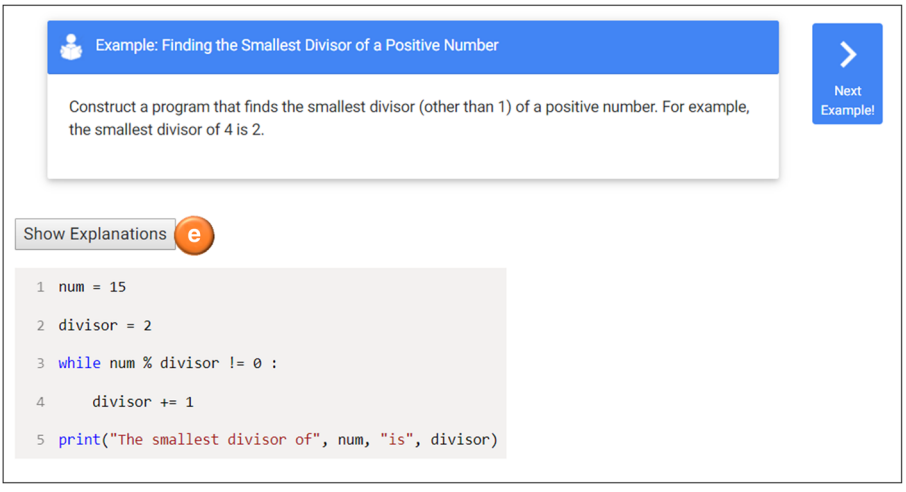

Fig. 5 The code-only mode of a worked example in the Control interface that presents worked examples focused on program construction skill. The student can click on link (**e**) to view the explanations

Study Procedure

The study was carried out in an introductory Python programming course. This course was a CS1 service course targeted to students of bachelor's degree programs in various engineering fields at a large research university. The students were not Computer Science majors, and most of them were likely to complete only one or two programming courses in their bachelor-level studies. The course consisted of lectures, nine exercise rounds, and an exam. To pass the course, the student had to solve enough problems (small Python coding exercises where the student mostly wrote the whole program alone) in each of the exercise Rounds 1–8 and pass the final test. To obtain a good grade, the student was also required to solve enough problems in Round 9, which contained coding exercises about object-oriented programming.

Potential participants included 723 undergraduate students who were enrolled in the course in the Fall semester of 2017. Students were randomly assigned to one of the Control and Experiment groups, explained in "Study Design". They were given one week to take an online pre-test (see "Pre- and Post-Tests"). In the second week of the class, the practice system with all practice content (see "Practice Content") was introduced and all students were provided with a link and an individual account to access the system. The use of the practice system was voluntary; however, the students were offered a few extra points to encourage them to explore the system and its practice content. More specifically, the students who explored at least eight examples and solved at least seven Parsons problems (see Fig. 6) were given 200 extra exercise points. The maximum number of exercise points the students could receive in this course by solving mandatory exercises (assignments) was 5820. In addition, the students who answered to the course feedback survey organized by the university were given 200 exercise points. Thus, the total number of exercise points a student could gather was 6220, from which 200 (about 3%) could be earned by using the practice system. Because the minimum number of the exercise points which the

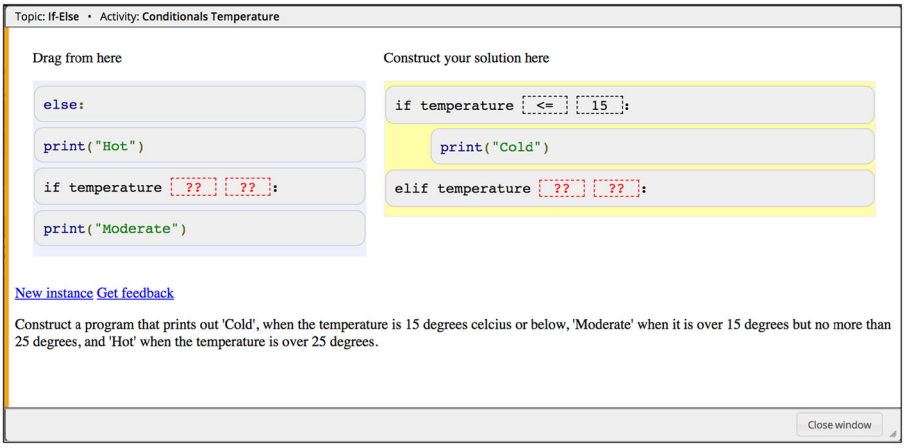

Fig. 6 An instance of a Parsons problem in the practice system. The student assembles the solution to the question (written at the bottom) by dragging lines of code to the right side

student needed for the best exercise grade was 5520, using the practice system was genuinely voluntary. The required minimum number of the examples and problems to receive those extra points was also much lower than the number of examples and problems available in the practice system (see "Practice Content"). Doing more than the required minimum was not incentivized by the awarding of any additional points.

A student could practice with the system up until the end of the semester, and at that point was given one week to respond to an online example evaluation survey (described in "Example Evaluation Survey") and take the online post-test, which was isomorphic to the pre-test. Completion of the pre-test, post-test, and survey was voluntary, too. Therefore, to increase students' participation, the pre-test was included in the first assignment, and students were offered a total of 4 extra final exam points for answering the post-test and the survey (2 points for each). The maximum number of exam points a student could achieve without those extra points was 96.

Materials

Practice Content

Practice content in the system included 52 example activities and 31 2D Parsons problems (Ihantola and Karavirta 2011). In this type of problem, the student was asked to construct the described program by putting the fragments in the correct order with the correct indentation (see Fig. 6). Feedback was shown upon student's request, which highlighted the correct and incorrect lines in the code.

Problem presentation was the same across the Control and Experimental groups, but examples were presented differently. In the Control group, examples were presented as textbook-style worked examples (as in Fig. 4), while in the Experimental group, examples were presented as PCEX activities (as in Figs. 1 and 2).

All practice content was organized into 15 topics that were ordered by increasing difficulty and presented through the Mastery Grids portal (Loboda et al. 2014) that used visual personal progress tracking (known as Open Student Modeling (OSM)) to engage students to work with the content. Each example activity started with a worked example and was followed by one to three similar examples (in the Control group) or challenges (in the Experimental group), and the median was one (1). In total, the example activities included 52 worked examples, collectively containing 531 line explanations, and 71 similar examples/challenges, which were presented after the worked examples in the Control/Experimental group.

Pre- and Post-Tests

The pre-test consisted of 10 questions that evaluated a student's prior knowledge of a subset of programming constructs that were covered in the course. The questions were designed to cover both simple and complex concepts in Python programming. The first five questions were multiple-choice questions that asked students to select the correct code snippet for each given task. The remaining five questions asked students to determine the correct order of the provided lines of code, in order to achieve a certain purpose. The lines were shuffled and included distractors. A

post-test was isomorphic to the pre-test, employing the same concepts but using different surface features. Cronbach's α between pre-test and post-test was .58 (indicating acceptable reliability). The maximum possible score on the pre/post-test was 10; one point was awarded for each question.

Metrics

We employed a variety of measures to compare the impact of the PCEX and textbook-style program construction examples on student's engagement and learning. Table 1 provides an overview of the metrics we used in our study. We grouped these into *engagement*, *performance*, and *learning* metrics.

Engagement Metrics

Behavioral engagement metrics focused on 1) work done on the examples and problems, and 2) overall system usage. The specific measures include the number of attempts, as well as the time on task for both examples and problems.

Each example activity consisted of subtasks that were similar code examples presented in different formats, depending on the group. In the Experimental group, the first subtask was a worked example and the next-to-last subtask was a challenge. In the Control group, all subtasks were worked examples presented in a static, textbook-style form. To measure the amount of work done on examples, we looked at the student's work on the first and next-to-last subtasks individually. Therefore, the collected data included the following measures for representing the work done on examples: number of first subtasks viewed, total time on first subtasks, number of next-to-last subtasks viewed, total time on next-to-last subtasks, and interactions with explanations. In the Control group, interactions with explanations indicated the number of times the student switched between modes with and without explanations. In the Experimental group, on the other hand, the interactions with explanations indicated the number of times the student clicked on the question mark symbol next to a line to view the explanation.

The data also included several measures related to overall system usage such as total time spent on the practice system, total attempts on activities, and number of sessions that the student practiced with the system.

Performance Metrics

We looked into coding performance within and outside of the practice system. The total correct attempts on Parsons problems and the number of distinct Parsons problems solved were measures of coding performance within the practice system. The measures of coding performance outside of the practice system included the total number of points that the student earned from coding assignments in the class and how early students submitted their coding assignments. It is not evident that earliness in assignment submission could be used as a standard measure of student's performance, as it could be a sign of better motivation or preparation. However, the study in Auvinen et al. (2018) has found some correlation between student's performance

Table 1 Overview of the metrics used to evaluate the PCEX examples

Aspect	Goal	Metric
Engagement	Work on examples	Number of first subtasks viewed
		Total time on first subtasks
		Number of next-to-last subtasks viewed
		Total time on next-to-last subtasks
		Interactions with explanations
	Work on problems	Total correct attempts on Parsons problems
		Distinct Parsons problems solved
		Total time on Parsons problems
	Overall system usage	Total practice time
		Total attempts on activities
		Number of practice sessions
Performance	Inside-system	Total correct attempts on Parsons problems
	problem-solving performance	Distinct Parsons problems solved
	Outside-system	Assignment points
	problem-solving performance	Early submission
Learning	Near transfer	Pre- and post-test
	Far transfer	Final exam
		• program comprehension questions
		• program construction questions
		(basic, complex)

and how early a student submits the assignment. As a result, we thought it would be interesting to see the impact of work on examples on early submission.

The assignment points were obtained from nine rounds of coding exercises during the course, producing a total with a minimum of 0 and maximum of 5,820 points. Each assignment could be completed during a certain time window during the term. Sometimes students spent a large amount of time completing the assignment and other times they were quicker (apparently better prepared from prior instruction). As a proxy measure for the degree of students' preparation, enabling them to complete their assignment sooner, we calculated the inverse of the median number of days that assignment was submitted after the assignment had been introduced.[1]

Learning Metrics

We measured student's learning by using the pre- and post-test (near transfer measure) and final exam score (far transfer measure). The final exam consisted of questions that assessed program comprehension and program construction skills. The

[1]We chose start date over due date as the time reference because some students submitted assignments after the due date was passed. Therefore, using the difference between the due date and the submission date would have made the interpretation of results more difficult, as some differences would be negative.

program construction questions were further grouped into two basic questions and one complex question. The first basic question asked about basic concepts in the course. The second basic question asked the student to complete a relatively simple task: writing a simple class and a main program that used that class. The complex question required a deeper understanding of loops, opening files, splitting strings, and exception handling, and was more difficult than the basic questions. We separated the grades for each group of exam questions. The maximum number of points that students could obtain was 21 on the program comprehension questions, 55 on the basic program construction questions, and 20 on the complex program question. The Cronbach's α between the three subcomponents in the exam was .67, which indicated an acceptable level of reliability.

Example Evaluation Survey

To collect students' feedback on the example activities, we administered a two-part survey related to system use and system impact. In the first part of the survey, students responded to questions 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 six follow-up items that focused on why the system was not used. Two of the items referred to a *bad system experience*, two emphasized *no help needed*, and two addressed *other reasons*, especially a poor introduction to the system and a lack of time to use the system. For this section of the survey and all questions in the second part of the survey, students were asked to respond using a 5-point Likert scale ranging from *Strongly Disagree* (1) to *Strongly Agree* (5).

The second part of the survey aimed to evaluate the impact of the example activities, focusing on only students who used the system. Following the suggestion in Kay and Knaack (2009) that identified key constructs required to evaluate a learning objective, we included three constructs: learning, quality, and engagement. Each construct had four items, with two negatively worded and two positively worded. For the learning construct, the items referred to a student's perception of how much they learned from the examples. For the quality construct, the items referred to the quality of the example activities. Finally, for the engagement construct, the items examined the level of student's involvement in the example activities.

Results

Students' Participation and Collected Data

Out of the 723 students enrolled in the class, only 202 used the system (i.e., had at least one attempt on an example or problem), 696 took the pre-test, 457 took the post-test, 447 took both the pre-test and post-test, and 456 answered the questionnaire. Among the students who used the system, 6 students had an extremely high pre-test score (i.e., 90 percent or above). We discarded the data of those 6 students since they had little to learn and were likely only participating for extra credit. The final

dataset included the data from 196 students: 118 students in the Control group and 78 students in the Experimental group. We used this data to perform our analysis of engagement and performance.

According to the learning gain data, we observed that some students had negative learning gains (minimum value was −.5), which likely reflects that a few students did not take the post-test seriously. More precisely, among 196 students who used the system, 49 earned fewer points in the post-test than in the pre-test, of which 32 were in the Control group and 17 were in the Experimental group. For only the learning analysis section, we excluded the group of students who had negative learning gains.[2] After discarding these students, we were left with data on 147 students (86 in the Control group, 61 in the Experimental) for the learning analysis. Note that there were no significant differences in the mean pre-test scores of both the Experimental ($M = 3.5, SD = 1.8$) and Control group ($M = 3.2, SD = 1.5$), $F(1, 116.08) = .77$, $p = 0.38$.

Prior to data analysis, we identified outliers in the collected data by using Tukey's box-plot method, which defines outliers as being outside the interval $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$, where Q stands for "quartile" and IQR stands for "interquartile range". We used winsorization to replace the outliers with less extreme values in the same direction (i.e., $Q1 - 1.5 \times IQR$ or $Q3 + 1.5 \times IQR$).

Finally, we examined the collected data to remove highly correlated measures. We observed that two of the measures related to the overall system usage, namely, total practice time and total attempts on activities, were highly correlated with each other ($\rho = 0.83$). Also, both measures were found to be highly correlated with the number of first subtasks viewed, and ρ was .80 for the total practice time and .99 for the total attempts on activities. As a result, we chose to use only the number of sessions as a measure of overall system usage. Similarly, the total correct attempts on Parsons problems was highly correlated with distinct Parsons problems solved ($\rho = 0.99$); therefore, we excluded the total number of Parsons problems solved from our analysis.

The following "Engagement Analysis", "Performance Analysis" and "Learning Analysis" present the results from the data analyses. Table 2 summarizes our main results by showing the hypotheses of classroom study (described in "Hypotheses"), the corresponding data analyses, and whether the hypotheses were confirmed by the data analyses. See the corresponding subsections for further details.

Engagement Analysis

Our first hypothesis ($H1$) was that PCEX would cause students to be more actively involved in PCEX worked examples than in the textbook-style worked examples. To test hypothesis $H1$, we used a one-way ANOVA analysis to compare the group means; for statistically significant cases, effect sizes using eta-squared (η^2) are presented in (Table 3). The metrics in Table 3 are ordered with a decreasing order of effect size. We used Cohen's rules of thumb for interpreting this data, with an effect

[2]Using all students' data showed the same pattern of results for this analysis.

Table 2 Summary of hypotheses and results of the classroom study (✓: confirmed hypothesis; ×: hypothesis that is not confirmed)

Hypotheses	Data analyses	Measures	Hypotheses confirmed? Group effect	Hypotheses confirmed? Interaction effect
H1 – PCEX vs. textbook-style examples: engagement	"Engagement Analysis"	time-on-task	✓	N/A
		interactions with explanations	×	N/A
H2 – PCEX vs. textbook-style examples: problem-solving	"Performance Analysis"	dist. Parsons problems solved	×	×
		assignment points	✓	×
		early submission	✓	✓
H3 – PCEX vs. textbook-style examples: learning	"Learning Analysis"	post-test	×	✓
		exam: code comprehension	×	×
		exam: basic code construction	×	×
		exam: complex code construction	✓	×

Table 3 Means (and SD) for engagement metrics in the Control and Experimental groups, along with inferential statistics and effect sizes, contrasting the groups

	Control (N=118) Mean ± SD	Experimental (N=78) Mean ± SD	One-way ANOVA	Effect size (η^2)
WORK ON EXAMPLES				
Total time on next-to-last subtasks (mins.)	8.3 ± 14.9	36.7 ± 29.5	$F(1, 103.12) = 61.91$, $p < .001^{***}$	.29 *L*
Interactions with explanations	30.0 ± 23.9	14.9 ± 24.0	$F(1, 164.18) = 18.74$, $p < .001^{***}$	.09 *M*
Total time on first subtasks (mins.)	2.1 ± 1.9	3.1 ± 2.3	$F(1, 144.79) = 10.54$, $p = .001^{**}$	.06 *M*
#first subtasks viewed	23.6 ± 18.9	19.0 ± 16.9	$F(1, 177.19) = 3.24$, $p = .073$.	.02 *S*
#next-to-last subtasks viewed	31.6 ± 27.2	24.5 ± 24.0	$F(1, 178.56) = 3.66$, $p = .057$.	.02 *S*
OVERALL SYSTEM USAGE				
Number of sessions	3.8 ± 3.0	5.1 ± 4.1	$F(1, 194) = 6.63$, $p = .011^{*}$	.03 *S*
WORK ON PROBLEMS				
Distinct Parsons problems solved	14.5 ± 11.9	12.3 ± 11.5	$F(1, 168.82) = 1.56$, $p = .213$	
Total time on Parsons problems (mins.)	31.0 ± 29.4	24.9 ± 26.5	$F(1, 176.56) = 2.29$, $p = .132$	

$^{***}p < .001$;$^{**}p < .01$; $^{*}p < .05$; . $p < .1$

Effect size: *S*=small; *M*=medium; *L*=large

size of 0.02 being considered "small" in magnitude, 0.06 being "medium", and 0.14 being "large".

We can see that among all the measures, the ones that are related to the worked examples display the largest differences between the two groups. The largest effect comes from working on the next-to-last subtasks. The mean of total time spent on the next-to-last subtasks was 4 times greater in the Experimental group, as compared to the Control group (36.7 vs. 8.3 mins.). This is expected because the next-to-last subtasks in the Experimental group required the student to solve the challenge, while the next-to-last subtasks in the Control group did not. The second largest effect was seen for working on the first subtasks. The total amount of time that students spent on the first subtasks was about 1.5 times greater in the Experimental group (3.1 minutes) than in the Control group (2.1 minutes).

We also observed that students had more interactions with explanations in the Control group than in the Experimental group. On average, the students in the Control group clicked on the show/hide explanations links 30 times, which was twice as high as in the Experimental group (14.9 times). We also found differences between the two groups in terms of the number of first subtasks and next-to-last subtasks that students viewed. Students in the Control group viewed, on average, more first subtasks and next-to-last subtasks; yet the differences reached only marginal significance and the effect size was small.

In addition to the differences between usage of activities, we also observed that the two groups were different in terms of number of practice sessions. The mean number of sessions differed by one across the two groups, with the Experimental group having more sessions ($M = 5.1, SD = 4.1$) than the Control group ($M = 3.8, SD = 3$). However, both the number of Parsons problems solved and the time spent on Parsons problems were not statistically different between the two groups.

In total, the ANOVA analyses of cognitive-behavioral engagement data partially supported hypothesis $H1$, favoring greater time-on-task but not more interactions with explanations. While students in the Control group viewed marginally more examples, they spent less time on the example subtasks. Students in the Experimental group, on the other hand, spent more time working with examples — about 1.5 more time on the first subtasks and 4 times more time on the next-to-last subtasks. The increase on time-on-task can be attributed to students becoming more involved when working with the PCEX examples than when working with the textbook-style worked examples.

Contrary to our hypothesis for greater interactions with the PCEX examples, we found that the Control group students used the "show/hide explanation" button more frequently than the Experimental group students clicked on example lines to view explanations. This difference, however, could be due to the way in which the explanations were presented in the textbook-style worked examples and PCEX examples. As mentioned earlier in "Control Condition Interface: Textbook-Style Worked Examples", explanations were shown by default in the textbook-style examples, to make the presentation of the worked examples similar to the programming textbooks. On the other hand, explanations were only available by taking action in the PCEX examples. Therefore, we hypothesize that the students viewing the textbook-style examples were influenced by needing a greater number of clicks to hide the

explanations compared to viewing the code alone, similar to the way in which it was presented to the Experimental group. Our data supports this hypothesis by showing that, on average, the median of clicks on the "hide explanation" button was twice that of the median of clicks on the "show explanation" button in an example (0.2 vs. 0.1).

Performance Analysis

Our second hypothesis ($H2$) was that working with PCEX would improve student's performance on program construction tasks more than that of the textbook-style worked examples. To test hypothesis $H2$, we ran a series of regression analyses—to examine the effect of the group, the amount of work on examples, and the interaction between the group and the amount of work on examples—for the student's coding performance, while controlling for prior learning, as indicated by the pre-test. That is, the independent variables were:

- *Group*: a dummy variable representing the group that student belonged to, with the Control group serving as the reference group factor;
- *WOE*: Work On Examples (*WOE*) is a continuous variable representing the combined work on the first subtasks and next-to-last subtasks in the example activities;
- *Pre-test*: a continuous variable representing the student's pre-test score; and
- The *Group* × *WOE* interaction: *Group* × *WOE* is the interaction between *Group* and *WOE* variables. This interaction means that the effect of *Group* on the performance measure is different for different values of work on examples. For example, a positive value for the effect of the interaction term would imply that the more work is done on examples, the more positive becomes the effect of the Experimental group on student's performance. All numeric independent variables (*Pre-test* and *WOE*) were mean-centered to reduce potential multicollinearity problems (Aiken et al. 1991).

The dependent variables were measures of coding performance, including:

- Distinct Parsons problems that the student solved;
- Total points that the student earned in the coding assignments; and
- Earliness of the student's submissions in the coding assignments.

The data that was used in the performance analysis was limited to the 194 students who used the system and who had taken the pre-test. Table 4 shows the results of the fitted models. The results revealed that *WOE* and *Pre-test* were positive predictors of distinct Parsons problems that student solved ($F(4, 189) = 217.2$, $p < .001$, $R^2 = .82$). Importantly, the effect of *WOE* on predicting distinct Parsons problems solved ($\beta = .88$) was about 14.7 times larger than the effect of the pre-test score ($\beta = .06$). We found no significant influence of the *Group* ($p = .371$) or the *Group* × *WOE* interaction ($p = .299$). This indicates that working on examples was associated with solving more Parsons problems correctly, regardless of the treatment group.

Group, *WOE*, and *Pre-test* were found to be predictors of assignments points, ($F(4, 189) = 5.54$, $p < .001$, $R^2 = .10$), since all were positively associated with assignment points. Among them, *Group* was the most influential ($\beta = .28$) and

Table 4 Regression results of Group, amount of work on examples (WOE), and the interaction of amount of work on examples with group (Group × WOE), predicting distinct Parsons problems solved, assignment points, and earliness of submission of coding assignments

	Dist. Parsons problems solved				Assignments points				Early submission			
Predictors	B	SE	β	R^2	B	SE	β	R^2	B	SE	β	R^2
Group	.67	.74	.06	.82	290.91	146.78	.28*	.10	.13	.04	.39 **	.22
WOE	.23	.01	.88 ***		6.07	2.00	.26 **		.00	.00	−.02	
Pre-test	.38	.21	.06 .		105.17	40.73	.18*		.05	.01	.29 ***	
Group × WOE	.02	.02	.07		−2.05	3.43	−.09		.004	.001	.59 ***	

*** $p < .001$;** $p < .01$; * $p < .05$; . $p < .1$

WOE was the second most influential predictor ($\beta = .26$). We found no significant effect for *Group* × *WOE* interaction ($p = .551$). These results suggest that although work on examples was generally helpful for getting more points in the coding assignments, it was overall more helpful to be in the Experimental group and practice with engaging examples.

The results of the regression analyses also showed that although *WOE* was not a significant overall predictor of earliness of submission ($p = .82$), *Group* and *Group* × *WOE* interaction were a predictor of early submission (Fig. 7). As the interaction plot shows, more activity with examples was associated with submitting assignments earlier in only the Experimental group. Furthermore, the interaction was found to be the most important predictor for earliness of submission ($\beta = .59$), with its effect being about two times greater than the effect of the pre-test score ($\beta = 0.29$).

In total, the multiple regression analyses supported hypothesis $H2$ for coding performance outside of the practice system, but not within the practice system. We found an overall positive effect in favor of the Experimental group on assignment points and earliness of submission. Being in the Experimental group and practicing with PCEX examples was associated with obtaining more points on coding assignments and also with submitting the assignments earlier. Furthermore, the interaction effect of the work with examples and the treatment group on earliness of submission was significant, which indicates that only more work on PCEX examples (and not textbook-style worked examples) led to earlier submission of assignments. Our analysis did not show any difference between the treatment groups in terms of performance within the practice system, though. More work with examples was associated with solving more Parsons problems, regardless of the treatment group; that is, textbook-style worked examples and PCEX examples were both helpful for improving student's performance on Parsons problems.

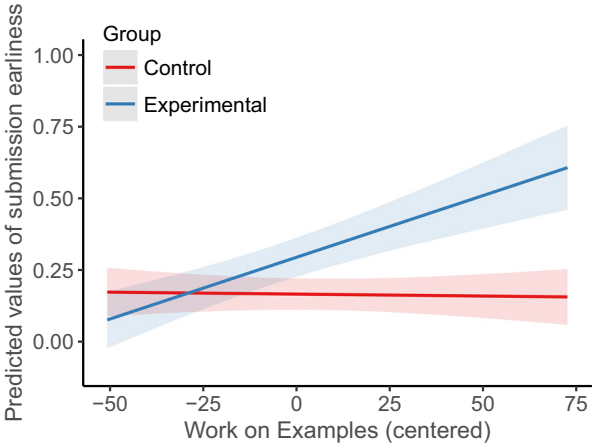

Fig. 7 Interaction between work on examples (WOE) and Group factor (Control/Experimental) for predicting the earliness of submissions for coding assignments. Notches indicate 95% confidence interval ranges

Learning Analysis

Our third hypothesis ($H3$) was that working with PCEX would improve learning outcomes more than working with the textbook-style worked examples. To test hypothesis $H3$, multiple regression analyses were performed to check the effect of the work on examples, the group, and the interaction between the work on examples and group on the post-test score and exam grade. The independent variables in all the regression models were similar to the independent variables in "Performance Analysis".

For predicting the post-test score, we used data of 194 students who used the system and who had taken the pre-test. For predicting the exam grade, this data was further limited to the 170 students who had taken the final exam. Table 5 shows the results of the regression model that tested the effect of *Group*, *WOE*, and *Group* × *WOE* interaction on the post-test score, controlling for the effect of the pre-test score. As expected, *Pre-test* was the most important predictor of students' post-test scores ($\beta = .42$). Neither *Group* ($p = .128$) nor *WOE* ($p = .677$) were accurate predictors of post-test scores, but *Group* × *WOE* was ($\beta = .29$); this finding suggests that the effect of the group depended on the amount of work on example (Fig. 8). As the figure shows, work with examples in the Experimental group increased the student's post-test scores more than work with examples in the Control group.

The results of the fitted models for predicting the exam grade are shown in Table 6. As the table shows, *WOE* positively predicts exam grade across all categories of questions; namely, program comprehension ($F(4, 165) = 3.98$, $p = .004$, $R^2 = .09$), basic program construction ($F(4, 165) = 2.56$, $p = .04$, $R^2 = .06$), and complex program construction ($F(4, 165) = 3.07$, $p = .018$, $R^2 = .07$). Also, the *Group* factor is a significant predictor for the exam grade on the complex question. The interaction *Group* × *WOE* is not statistically significant for any exam category. For the complex program construction questions, *Group* ($\beta = .28$) and *WOE* ($\beta = .23$) are the first and second most important predictors, respectively, and the effect of each was about two times greater than the effect of the *Pre-test* ($\beta = .12$).

In total, the multiple regression analyses of the learning outcomes supports hypothesis $H3$, which demonstrates a significant positive interaction between work with examples and treatment group on post-test (near transfer test) and a marginally positive effect of the treatment group on the complex program construction question in the final exam (far transfer test). More specifically, more work with only the PCEX examples (and not the textbook-style worked examples) was associated with a higher post-test score. Additionally, being in the Experimental group and practicing

Table 5 Regression results of predicted post-test score while controlling for the pre-test score

Predictors	B	SE	β	R^2
Group	.32	.21	.22	.22
WOE	.0	.0	.04	
Pre-test	.35	.06	.42***	
Group × WOE	.01	.0	.29*	

$^{***}p < .001; ^{**}p < .01; .\ p < .1$

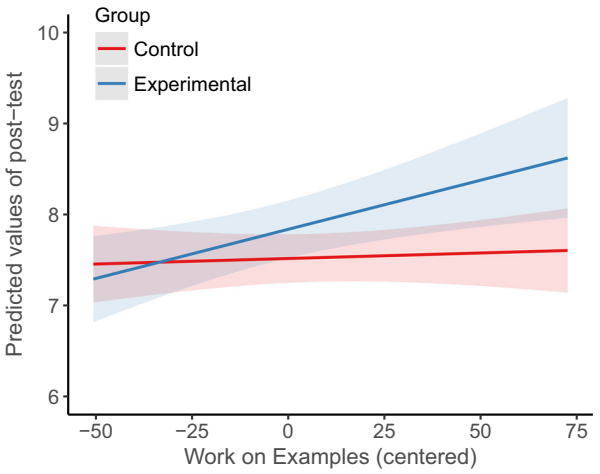

Fig. 8 Interaction between work on examples (WOE) and Group factor (Control/Experimental) for predicting the post-test score. Notches indicate 95% confidence interval ranges

with PCEX examples was marginally associated with obtaining a higher grade in the complex program construction question of the final exam. We found no differences between the PCEX and textbook-style worked examples on other questions in the final exam. Work with examples in both groups was found to be positively associated with obtaining a higher grade in the program comprehension questions and basic program construction questions of the final exam.

Survey Analysis

Before analyzing the survey group differences, we assessed each construct's reliability using Cronbach's α. We dropped two items from the engagement construct and

Table 6 Regression results of group, amount of work on examples (WOE), and the interaction of amount of work on examples with group predicting exam grade on the program comprehension and construction questions

					Program Construction							
	Program Comprehension				Basic				Complex			
Predictors	B	SE	β	R^2	B	SE	β	R^2	B	SE	β	R^2
Group	−.66	.75	−.13	.09	1.03	1.68	.09	.06	1.22	.66	.28.	.07
WOE	.02	.01	.20*		.06	.02	.23*		.02	.01	.23*	
Pre-test	.43	.21	.16*		.94	.47	.15*		.31	.18	.12.	
Group × WOE	.01	.02	.10		−.04	.04	−.15		−.01	.02	−.12	

*$p < .05$; . $p < .1$

one item from the quality construct because their item-construct correlations were lower than the recommended value of .30. Additionally, we checked whether the internal consistency could improve if any of the items within a construct were deleted. We discarded one item in the quality construct because the removal of that item increased the internal consistency among the items of the construct and improved the α from .7 to .73. No item was discarded from the learning and engagement construct, as all items had acceptable internal consistency with the other items within that construct. The α was .74 for the learning construct and .7 for the engagement construct. After this step, all three constructs appeared to be sufficiently reliable for assessing the value of the examples, with α values exceeding the suggested minimum acceptable α coefficient of .50 (Nunnally 1978).

The survey was available for all students who enrolled in the course. As mentioned in "Example Evaluation Survey", only students who used the system were able to respond to the second part of the survey that sought to evaluate the examples. Out of the 456 students who completed the survey, 15% ($N = 67$) used the system more than 10 times, 14% ($N = 62$) used the system between 5 and 10 times, 26% ($N = 120$) used the system less than 5 times, and 45% ($N = 207$) did not use the system at all. The mean and standard deviation for the survey items (except the items that were discarded during the reliability analysis) are shown in Table 7. Overall, students mostly agreed that they did not use the system due to preferring other resources and materials, not feeling the need for additional help, and a lack of time. Students disagreed with items that suggested other reasons for low/zero usage of the practice system, including the items that referred to having a bad system experience.

Overall, about 60% of the students agreed with each of the items in the quality construct, while less than half of the students agreed with each of the items related to the learning construct (the agreement level ranged from 30% to 45%). Overall agreement with each of the items in the engagement construct was low (below 30%). As a result, many students did not perceive the examples to be engaging. In both groups, students were most positive about the quality of examples and the least positive about overall engagement with the examples. The Kruskal-Wallis test showed no significant differences in the the mean ratings of the study groups in the quality construct ($\chi^2(1) = .67$, $p = .41$), learning construct ($\chi^2(1) = .26$, $p = .61$), or the engagement construct ($\chi^2(1) = .32$, $p = .57$). The pattern of results did not change when we excluded data of students who used the system fewer than 10 times or fewer than 5 times.

Summary and Discussion

Summary

This work describes the classroom study that was conducted to address RQ1, RQ2, and RQ3 (described in "Introduction"), in order to measure the effect of PCEX examples on student's engagement, problem-solving performance, and learning, as compared to non-interactive textbook-style examples. This section summarizes and discusses our findings from the study.

Table 7 Mean and standard deviation of the responses for the classroom study survey. The response options represented a 5-point Likert scale, ranging from 1 (Strongly Disagree) to 5 (Strongly Agree)

Item	Mean (SD)
REASONS FOR LOW/ZERO USAGE	
I preferred to use other resources and material to learn Python	4.0 (0.9)
I was doing well in class without the system and did not need any extra help	3.9 (1.0)
I did not have enough time to use the system	3.4 (1.1)
The system was not introduced properly in class	3.0 (1.0)
I didn't think the system can help me to better master Python	3.0 (0.9)
The user interface was too confusing to use	2.9 (0.8)
EXAMPLE EVALUATION	
Quality	
The explanations in the examples were easy to follow	3.6 (0.8)
The explanations in the examples were not hard to understand[a]	3.5 (0.9)
Learning	
The explanations in the examples helped me to better understand the Python programming concepts[a]	3.3 (0.8)
Working with the examples helped me learn Python	3.3 (0.8)
Exploring similar examples helped me learn Python	3.2 (0.7)
The examples helped me in solving Python exercises in this class[a]	3.0 (0.9)
Engagement	
I tried hard to understand the examples	2.9 (1.0)
I did not skim-read the examples[a]	2.6 (1.0)

[a]A reverse-coded item

Overall Effects on Engagement

Students were more engaged in the work with PCEX than with the textbook-style worked examples. The largest effect comes from the work on the next-to-last subtasks. The mean of total time spent on the next-to-last subtasks was 4 times greater in the Experimental group compared to the Control group. The second largest effect was for the work on the first subtasks. The total time that a student spent on the first subtasks was about 1.5 times higher in the Experimental group than in the Control group. We also observed that the mean interaction with explanations was low in both groups, which suggests that the interactivity element implemented in the first subtasks in the Experimental group was not as engaging as that in the next-to-last subtasks (i.e., challenges). We also observed that students in Experimental group had, on average, more practice sessions. Although the effect size was small, it suggests that students in the Experimental group were more interested in returning to the system for practice. We could attribute this to PCEX, since the only difference between the groups was in how examples were presented in the system.

Overall Effects on Problem-Solving Performance

We found that working on examples and having higher pre-test scores were associated with solving more Parsons problems correctly, regardless of the treatment group. Among these two predictors, the effect of the work on examples on predicting distinct Parsons problems solved was about 14.7 times larger than the effect of the pre-test score. We also found that work on examples, group, and pre-test were all positive predictors of assignment points. Notably, group and work on examples were found to be the most influential predictors, with higher predictive power than the pre-test scores. We also found that the effect of the group on earliness of submission depended on the amount of work on examples. More work on the examples in the Experimental group was associated with submitting assignments earlier. Furthermore, the interaction was found to be the most important predictor for the earliness of submission; its effect was about two times greater than that of the pre-test score.

Overall Effects on Learning Outcomes

For predicting post-test scores, as expected, pre-test scores had the most predictive power. Yet, we observed that work with examples in the Experimental group increased the students' post-test scores more than the work with examples in the Control group. This difference increased as the amount of work increased in the Experimental group. Furthermore, work on both examples and pre-test positively predicted exam grade across all categories of questions. Work on examples became a more important predictor than pre-test scores as our analysis moved from program comprehension to basic program construction and then to complex program construction questions in the exam. For the complex program construction questions, group was also a positive predictor, and the effect of the group and work on examples was about two times greater than that of the pre-test scores.

Past research on engagement (Skinner et al. 2009; Christenson et al. 2012; Reeve 2013; Reeve and Tseng 2011) hints that the observed increase in learning outcomes is caused by both a higher level of cognitive engagement with the new examples (which was our design goal) and a considerably higher level of behavioral engagement (which was directly observed). Our study does not differentiate between these two factors. However, even if the larger share of performance increase was caused by the observed behavioral engagement increase (i.e., time and efforts on task), it is still a highly desired and important outcome in our explored context: the work with voluntary content. In the context of modern e-learning, more and more learning content is released to the students in a voluntary form; i.e., the students are encouraged, but not required, to work with such content and no grade points are offered for doing so. Moreover, a considerable fraction of modern interactive content (simulations, animations, worked examples) are not designed to assess student's performance and are difficult to grade objectively. In this context, not the effectiveness, but the simple amount of work becomes the most critical factor in ensuring the impact of this content on student's learning. While novel types of content, such as animated program examples, are usually proven to be effective in lab studies where the use of this content is required, a number of classroom studies confirmed that this content has low

to zero usage when it is offered in a voluntary form (Naps et al. 2002). Thus, the very increase of voluntary work with learning content when it is offered in a more engaging and interactive form is an important desired outcome.

Students' Feedback

Survey results showed that students were positive toward the quality of explanation, almost neutral toward the helpfulness of the examples for learning, and did not perceive examples to be engaging.

Discussion

Our learning technology tool, PCEX, was designed to replicate and expand (to another domain) prior research on worked examples in the domains of math and science (e.g., Atkinson et al. (2000), Chi et al. (1989), and Sweller and Cooper (1985)). It was also designed to help students acquire program construction skills by presenting examples in an engaging fashion. Our findings from the classroom study support the positive impact of PCEX examples on student's engagement, problem-solving performance, and learning. Our findings reconfirm the results of previous work in the domains of math and science (Atkinson and Renkl 2007) and program behaviour examples (Hundhausen et al. 2002; Evans and Gibbons 2007; Naps 2005) which showed that interactive worked examples help achieve better learning outcomes. However, some aspects of our findings need to be discussed further:

First, while the measure of earliness of submission could be an indicator of how students were prepared better or worse to submit the code, we also acknowledge that submitting coding assignments earlier could have been due to better time management skills or other factors related to a student's self-regulation skills.

Second, on one hand, hypothesis $H1$ states that engaging examples increase student's engagement (e.g., time on task) while working with the examples. On the other hand, Hypotheses $H2$ and $H3$ state that engaging examples lead to better performance in coding and learning outcomes, respectively. We acknowledge that $H2$ and $H3$ are likely results of $H1$. In general, time on task is positively correlated with learning and better performance, but the main challenge is how to get the student motivated to spend more time. The use of engaging examples, which we proposed in this work, is a solution to this challenge. Our results showed that students spent more time on the PCEX examples because they were engaged more and, as a result, learned more and obtained better problem-solving performance.

Third, we observed from the survey responses that students in both groups were neutral toward learning from examples and had a negative opinion about being engaged by the examples. These observations contradict what we have found from our engagement, learning, and performance analyses, because results showed that PCEX examples involved students in working more with examples; thus, increasing the time spent on task. In a similar fashion, work with PCEX examples improved both student's learning and coding skills.

The difference in the quantitative and survey analysis implies that, first, PCEX examples engaged students and improved their learning — but students did not

realize that they were engaged and improving their skills. Second, lower ratings might mean that students had higher expectations for the learning tool in the context of our study. Third, it might be that students perceived engagement items in the survey differently than we expected they would. Another reason might be that when students were questioned about how engaging the PCEX examples were (survey), they were comparing it to game apps they play, but when they began to study programming, their actual engagement with the system (log-based data) was reflecting how enthusiastic they felt about interactive PCEX examples, as compared to studying a flat, static book (or the tutoring equivalent of a flat, static book).

Finally, we acknowledge that there is a blurred line between PCEX examples and problems due to the dual nature of "engaging" features within tutoring systems. In general, any engagement asking for additional student's action will cause the student to move from examples to problem-solving. Yet, we find *examples* to be a more appropriate category for our developed learning tool rather than *problems*, because the "engaging" features that we are using in the PCEX examples have been used in previous learning tools that are known as examples. In particular, in the domain of CS education, animated examples engage the student by asking a question or allowing the student to change the input used by the example. Other aspects that shaped our study include similarities to previous example research in the domains of math and science, such as missing steps that the student has to fill in, or self-explanation prompts that the student has to answer.

Limitations and Future Work

This work can be extended in several ways. The first direction for future work would be to improve the PCEX interface features, while the second direction for future work would be to extend the study conducted in this work that assessed the impact of the examples. The last direction for future work would be to share the PCEX examples with a broader audience, as well as to conduct more studies that connect our work with previous work on program behavior examples. The following subsections discuss these and other directions for future work.

Example Design

In the current design, PCEX combines explorability and challenge in the same interface, which makes it impossible to separately assess the value of explorability and challenges. In the future, it would be good to conduct a study to evaluate the combined and separate impact of explorability and challenge in the PCEX activities. Also, the interface of worked examples does not have any features to highlight which concepts or code segments are important to be studied for each block of program code. This is not an issue for small program coding, but the student may easily get lost when the program code has many lines of code. Future work may explore possible ways to emphasize important code segments or motivate the student to think more deeply about the code. One approach would be to use self-explanation prompts in the worked examples. Future research should study how and when to present the self-explanation prompts to enhance learning from examples. Finally, our studies showed

that students rarely accessed available hints and explanations, and that it was not clear why these features were not used extensively. Thus, future research needs to run more usability studies for understanding how to improve the design of these features.

The Study Assessing Examples

First, the usage of the practice system was voluntary and, as a result, many of the students did not use the system. For example, only 200 (28%) students used the system in the classroom study. It's not clear how this bias affected this group of students. This limitation stems from the voluntary nature of practice, which might appeal to certain types of students. Although self-selection limits us from having a reliable, fully controlled study, it also helps us to conduct studies that are closer to the natural context of learning to program. Specifically, students who seek help for learning to program have access to abundant resources for learning programming, including online tutors such as the Python tutor, programming platforms such as CodeWorkOut, CloudCoder, and CodingBat, and programming MOOCs and video tutorials that are available on YouTube. The non-mandatory design of the classroom study in this work enabled us to investigate the impact of the proposed learning tool in natural context. Future work may investigate what happens when the system is offered in a mandatory way to see if mandatory work within the practice system would have positive effects on students who would not be using it otherwise.

Second, due to the loosely controlled nature of the classroom study, students may have learned the skills that we controlled for by using resources outside the practice system. This is a factor that we can't control for in our analyses. Future work may investigate the impact of the system in a more controlled way, perhaps by distributing the practice across multiple lab sessions and assessing changes in student's knowledge both before and after each practice session.

Although the classroom study was conducted in a rather large course, it was limited to a specific population. The target course in the study was presented as a mandatory course for engineering students. The participants were mainly studying electrical or civil engineering with a small number of students from other engineering programs. For most students, except for electrical engineering students, this was the only compulsory programming course in their curriculum. Our experience has been that there is a considerable portion of such non-CS students who are not highly motivated to learn to program, especially when compared with computer science (CS) students. Thus, our results from the classroom study may not generalize well for CS major students elsewhere, but may better generalize for CS minors or non-CS majors. Another limitation in the classroom study was that the Control and Experimental groups had different numbers of students using the system. This is a limitation of our analysis and our results could have been influenced by these groups having unmatched numbers. Therefore, a similar study should be conducted in the future with CS majors to validate our observations in the classroom study.

Future studies should also conduct individual interviews with students to understand the possible reasons for reporting lower engagement in the survey, and also to discuss options for enhancing engaging features in PCEX.

Other Directions

We plan to connect our work to previous work on program behavior tools. In particular, we plan to study the impact of integrating program behavior and program construction learning activities when they are offered together in a practice system. We also plan to build a repository of PCEX examples to make them available to the public. Finally, we plan to develop an open-source authoring tool to allow instructors and researchers to create and share their own PCEX examples.

References

Aiken, L.S., West, S.G., Reno, R.R. (1991). Multiple regression: Testing and interpreting interactions. Sage.

Atkinson, R.K., & Renkl, A. (2007). Interactive example-based learning environments: Using interactive elements to encourage effective processing of worked examples. *Educational Psychology Review*, *19*(3), 375–386.

Atkinson, R.K., Derry, S.J., Renkl, A., Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of educational research*, *70*(2), 181–214.

Auvinen, T., Hakulinen, L., Malmi, L. (2018). Increasing students' awareness of their behavior in online learning environments with visualizations and achievement badges. *IEEE Transactions on Learning Technologies*, *8*(3), 261–273.

Bathgate, M., & Schunn, C. (2017). The psychological characteristics of experiences that influence science motivation and content knowledge. *International Journal of Science Education*, *39*(17), 2402–2432.

Bloom, S.B. (1956). *Taxonomy of educational objectives, handbook I: The cognitive domain*. New York: David McKay Co Inc.

Brna, P. (1998). Searching for examples with a programming techniques editor. *Journal of Computing and Information Technology*, *6*(1), 13–26.

Brusilovsky, P. (1994). Explanatory visualization in an educational programming environment: connecting examples with general knowledge. In: 4th international conference on human-computer interaction, EWHCI'94, vol. 876 of Lecture Notes in Computer Science, pp. 202–212. Springer-Verlag.

Brusilovsky, P., & Yudelson, M.V. (2008). From webex to navex: Interactive access to annotated program examples. *Proceedings of the IEEE*, *96*(6), 990–999.

Brusilovsky, P., Yudelson, M., Hsiao, I.-H. (2009). Problem solving examples as first class objects in educational digital libraries: Three obstacles to overcome. *Journal of Educational Multimedia and Hypermedia*, *18*(3), 267–288.

Byrne, M.D., Catrambone, R., Stasko, J.T. (1999). Evaluating animations as student aids in learning computer algorithms. *Computers & education*, *33*(4), 253–278.

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General*, *127*(4), 355–376.

Chen, X., Mitrovic, A., Matthews, M. (2019a). Learning from worked examples, erroneous examples and problem solving: Towards adaptive selection of learning activities. IEEE Transactions on Learning Technologies, pages 1–1.

Chen, X., Mitrovic, A., Matthews, M. (2019b). Investigating the effect of agency on learning from worked examples, erroneous examples and problem solving. International Journal of Artificial Intelligence in Education.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, *13*(2), 145–182.

Christenson, S.L., Reschly, A.L., Wylie, C. (2012). Handbook of research on student engagement. Springer Science & Business Media.

Cooper, S., Dann, W., Pausch, R. (2003). Teaching objects-first in introductory computer science. In: ACM SIGCSE Bulletin, vol. 35, pp. 191–195. ACM.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., Cooper, S. (2012). Mediated transfer: Alice 3 to java. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 141–146. ACM.

Davidovic, A., Warren, J., Trichina, E. (2003). Learning benefits of structural example-based adaptive tutoring systems. *IEEE Transactions on Education*, *46*(2), 241–251.

Ericson, B.J., Guzdial, M.J., Morrison, B.B. (2015). Analysis of interactive features designed to enhance learning in an ebook. In: Proceedings of the 11th annual international conference on international computing education research, pp. 169–178 ACM.

Ericson, B.J., Margulieux, L.E., Rick, J. (2017). Solving parsons problems versus fixing and writing code. In: Proceedings of the 17th Koli calling conference on computing education research, pp. 20–29. ACM.

Esteves, M., & Mendes, A. (2003). Oop-anim, a system to support learning of basic object oriented programming concepts. In: Proceedings of compsystech'2003-international conference on computer systems and technologies, Sofia. Bulgaria.

Evans, C., & Gibbons, N.J. (2007). The interactivity effect in multimedia learning. *Computers & Education*, *49*(4), 1147–1160.

Fabic, G.V.F., Mitrovic, A., Neshatian, K. (2019). Evaluation of parsons problems with menu-based self-explanation prompts in a mobile python tutor. International Journal of Artificial Intelligence in Education. ISSN 1560-4306.

Getao, K.W. (1990). An environment to support the use of program examples while learning to program in lisp. In: Proceedings of the IFIP TC13 3rd international conference on human-computer interaction, pp. 1015–1016. North-Holland Publishing Co.

Hansen, S.R., Narayanan, N.H., Schrimpsher, D. (2000). Helping learners visualize and comprehend algorithms. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 2(1).

Harvey, B., & Mönig, J. (2010). Bringing "no ceiling" to scratch: Can one language serve kids and computer scientists. Proc Constructionism.

Hundhausen, C.D., Douglas, S.A., Stasko, J.T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, *13*(3), 259–290.

Ihantola, P., & Karavirta, V. (2011). Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations. *Journal of Information Technology Education: Innovations in Practice*, *10*, 1–14.

Kalyuga, S., Chandler, P., Sweller, J. (2000). Incorporating learner experience into the design of multimedia instruction. *Journal of Educational Psychology*, *92*(1), 126–136.

Kalyuga, S., Chandler, P., Tuovinen, J., Sweller, J. (2001). When problem solving is superior to studying worked examples. *Journal of Educational Psychology*, *93*(3), 579–588.

Kalyuga, S., Ayres, P., Chandler, P., Sweller, J. (2003). The expertise reversal effect. *Educational Psychologist*, *38*(1), 23–31.

Kay, R.H., & Knaack, L. (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), 147–168.

Khandwala, K., & Guo, P.J. (2018). Codemotion: expanding the design space of learner interactions with computer programming tutorial videos. In: Proceedings of the Fifth Annual ACM Conference on Learning at Scale, pp. 57, 1–57, vol. 10.

Lawrence, A.W. (1993). *Empirical studies of the value of algorithm animation in algorithm understanding*. PhD thesis: Georgia Institute of Technology.

Lieberman, H. (1987). An example-based environment for beginning programmers. In *Artificial intelligence and education* (pp. 135–151). Norwood: Ablex Publishing.

Linn, MC., & Clancey, M.J. (1992). The case for case studies of programming problems. *Communications of the ACM*, *35*(3), 121–132.

Loboda, T., Guerra, J., Hosseini, R., Brusilovsky, P. (2014). Mastery grids: An open source social educational progress visualization. In de Freitas, S, Rensing, C., Muñoz Merino, P.J., Ley, T. (Eds.) *9th European conference on technology enhanced learning (EC-TEL 2014), vol. 8719 of lecture notes in computer science* (pp. 235–248).

Loboda, T.D., & Brusilovsky, P. (2010). User-adaptive explanatory program visualization: Evaluation and insights from eye movements. *User Modeling and User-Adapted Interaction*, *20*(3), 191–226.

McLaren, BM., Adams, D., Durkin, K., Goguadze, G., Mayer, RE., Rittle-Johnson, B., Sosnovsky, S., Isotani, S., van Velsen, M. (2012). To err is human, to explain and correct is divine: A study of interactive erroneous examples with middle school math students. In: 7th European conference

on technology enhanced learning (EC-TEL 2012), vol. 7563 of lecture notes in computer science, pp. 222–235.

Miller, B.N., & Ranum, D.L. (2012). Beyond pdf and epub: toward an interactive textbook. In: Proceedings of the 17th ACM annual conference on innovation and technology in computer science education, pp. 150–155. ACM.

Miyadera, Y., Kurasawa, K., Nakamura, S., Yonezawa, N., Yokoyama, S. (2007). A real-time monitoring system for programming education using a generator of program animation systems. *JCP*, *2*(3), 12–20.

Morrison, B.B., Margulieux, L.E., Ericson, B., Guzdial, M. (2016). Subgoals help students solve parsons problems. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pages 42–47 ACM.

Myller, N. (2006). Automatic prediction question generation during program visualization. In: Proceedings of the 4th program visualization workshop.

Najar, A.S., Mitrovic, A., McLaren, B.M. (2016). Learning with intelligent tutors and worked examples: Selecting learning activities adaptively leads to better learning outcomes than a fixed curriculum. *User Modeling and User-Adapted Interaction*, *26*(5), 459–491. ISSN 1573-1391.

Naps, T.L. (2005). Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, *25*(5), 49–55.

Naps, T.L., Eagan, J.R., Norton, L.L. (2000). Jhavé – an environment to actively engage students in web-based algorithm visualizations. In: ACM SIGCSE bulletin, vol. 32, pp. 109–113. ACM.

Naps, TL., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J.Á. (2002). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE bulletin*, *35*, 131–152.

Nokes-Malach, T.J., VanLehn, K., Belenky, D.M., Lichtenstein, M., Cox, G. (2013). Coordinating principles and examples through analogy and self-explanation. *European Journal of Psychology of Education*, *28*(4), 1237–1263.

Nunnally, J.C. (1978). Psychometric Ttheory: 2d Ed. McGraw-Hill.

Paas, F.G.W.C., & Van Merriënboer, J.J.G. (1994). Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of educational psychology*, *86*(1), 122–133.

Park, J., Park, Y.H., Kim, J., Cha, J., Kim, S., Alice, O.H. (2018). Elicast: embedding interactive exercises in instructional programming screencasts. In *Proceedings of the 5th annual ACM conference on learning at scale, pp. 58*, (Vol. 10 pp. 1–58).

Parsons, D., & Haden, P. (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, pp. 157–163. Australian Computer Society Inc.

Pirolli, P.L., & Anderson, J.R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue canadienne de psychologie*, *39*(2), 240–272.

Reeve, J. (2013). How students create motivationally supportive learning environments for themselves: The concept of agentic engagement. *Journal of Educational Psychology*, *105*(3), 579–595.

Reeve, J., & Tseng, C.-M. (2011). Agency as a fourth aspect of students' engagement during learning activities. *Contemporary Educational Psychology*, *36*(4), 257–267.

Renkl, A. (1997). Learning from worked-out examples: A study on individual differences. *Cognitive science*, *21*(1), 1–29.

Renkl, A., & Atkinson, R. (2007). An example order for cognitive skill acquisition. Oxford University Press.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Rivers, K. (2017). *Automated data-driven hint generation for learning programming*. PhD thesis: Carnegie Mellon University.

Sajaniemi, J., & Kuittinen, M. (2003). Program animation based on the roles of variables. In: Proceedings of the ACM symposium on Software visualization, pp. 7–ff. ACM.

Salden, R.J.C.M., Aleven, V., Schwonke, R., Renkl, A. (2010a). The expertise reversal effect and worked examples in tutored problem solving. *Instructional Science*, *38*(3), 289–307. ISSN 1573-1952.

Salden, R.J.C.M., Koedinger, K.R., Renkl, A., Aleven, V., McLaren, B.M. (2010b). Accounting for beneficial effects of worked examples in tutored problem solving. *Educational Psychology Review*, *22*(4), 379–392.

Sears, A., & Wolfe, R. (1995). Visual analysis: Adding breadth to a computer graphics course. In: ACM SIGCSE Bulletin, vol. 27, pp. 195–198. ACM.

Sharrock, R., Hamonic, E., Hiron, M., Carlier, S. (2017). Codecast: An innovative technology to facilitate teaching and learning computer programming in a c language online course. In: Proceedings of the 4th ACM conference on learning at scale, pp. 147–148. ACM.

Sirkiä, T. (2013). A javascript library for visualizing program execution. In: Proceedings of the 13th Koli calling international conference on computing education research, pp. 189–190. ACM.

Skinner, E.A., Kindermann, T.A., Furrer, C.J. (2009). A motivational perspective on engagement and disaffection: Conceptualization and assessment of children's behavioral and emotional participation in academic activities in the classroom. *Educational and Psychological Measurement*, *69*(3), 493–525.

Sorva, J., Karavirta, V., Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 15:1–15:64.

Sweller, J., & Cooper, G.A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and instruction*, *2*(1), 59–89.

Sweller, J., Van Merrienboer, J.J.G., Paas, F.G.W.C. (1998). Cognitive architecture and instructional design. *Educational psychology review*, *10*(3), 251–296.

Trafton, J.G., & Reiser, B.J. (1993). The contributions of studying examples and solving problems to skill acquisition. In: Proceedings of the 15th annual conference of the cognitive science society, pp. 1017–1022 ACM.

Walker, C.O., Greene, B.A., Mansell, R.A. (2006). Identification with academics, intrinsic/extrinsic motivation, and self-efficacy as predictors of cognitive engagement. *Learning and individual differences*, *16*(1), 1–12.

Ward, M., & Sweller, J. (1990). Structuring effective worked examples. *Cognition and instruction*, *7*(1), 1–39.

Weber, G. (1996). Individual selection of examples in an intelligent learning environment. *Journal of Interactive Learning Research*, *7*(1), 3–31.

Weber, G., & Brusilovsky, P. (2001). Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education (IJAIED)*, *12*, 351–384.

Weber, G., & Mollenberg, A. (1994). Elm-pe: A knowledge-based programming environment for learning lisp. In: Proceedings of ED-MEDIA 1994, pp. 557–562 ERIC.

Yudelson, M., Brusilovsky, P., Zadorozhny, V. (2007). A user modeling server for contemporary adaptive hypermedia: An evaluation of push approach to evidence propagation. In Conati, C., McCoy, K., Paliouras, G. (Eds.) *11th International Conference on User Modeling, UM 2007, vol. 4511 of Lecture Notes in Computer Science, pp/ 27–36. Springer Verlag*.

Zhi, R., Price, T.W., Marwan, S., Milliken, A., Barnes, T., Chi, M. (2019). Exploring the impact of worked examples in a novice programming environment. In: Proceedings of the 50th ACM technical symposium on computer science education, SIGCSE '19, pp. 98–104. ACM.

Affiliations

Roya Hosseini[1] · Kamil Akhuseyinoglu[1] · Peter Brusilovsky[1] · Lauri Malmi[2] · Kerttu Pollari-Malmi[2] · Christian Schunn[1] · Teemu Sirkiä[2]

Kamil Akhuseyinoglu
kaa108@pitt.edu

Peter Brusilovsky
peterb@pitt.edu

Lauri Malmi
lauri.malmi@aalto.fi

Kerttu Pollari-Malmi
kerttu.pollari-malmi@aalto.fi

Christian Schunn
schunn@pitt.edu

Teemu Sirkiä
teemu.sirkia@aalto.fi

1 University of Pittsburgh, Pittsburgh, PA, USA

2 Aalto University, Espoo, Finland