PROGRAM CONSTRUCTION EXAMPLES IN COMPUTER SCIENCE EDUCATION: FROM STATIC TEXT TO ADAPTIVE AND ENGAGING LEARNING TECHNOLOGY

by

Roya Hosseini

B.S. in IT Engineering, Shiraz Univ. of Technology, 2009
M.S. in IT Engineering (E-Commerce),
Amirkabir Univ. of Technology, 2012
M.S. in Intelligent Systems, Univ. of Pittsburgh, 2015

Submitted to the Graduate Faculty of
the School of Computing and Information
in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Roya Hosseini

It was defended on

July 24th 2018

and approved by

Peter Brusilovsky, PhD, Professor

Christian D. Schunn, PhD, Professor

Diane Litman, PhD, Professor

Vincent Aleven, PhD, Associate Professor

Dissertation Director: Peter Brusilovsky, PhD, Professor

PROGRAM CONSTRUCTION EXAMPLES IN COMPUTER SCIENCE EDUCATION: FROM STATIC TEXT TO ADAPTIVE AND ENGAGING LEARNING TECHNOLOGY

Roya Hosseini, PhD

University of Pittsburgh, 2018

My dissertation is situated in the field of computer science education research, specifically, the learning and teaching of programming. This is a critical area to be studied, since, primarily, learning to program is difficult, but also, the need for programming knowledge and skills is growing, now more than ever. This research is particularly focused on how to support a student's acquisition of program construction skills through worked examples, one of the best practices for acquiring cognitive skills in STEM areas.

While learning from examples is superior to problem-solving for novices, it is not recommended for intermediate learners with sufficient knowledge, who require more attention to problem-solving. Thus, it is critical for example-based learning environments to adapt the amount and type of assistance given to the student's needs. This important matter has only recently received attention in a few select STEM areas and is still unexplored in the programming domain. The learning technologies used in programming courses mostly focus on supporting student problem-solving activities and, with few exceptions, examples are mostly absent or presented in a static, non-engaging form.

To fill existing gaps in the area of learning from programming examples, my dissertation explores a new genre of worked examples that are both adaptive and engaging, to support students in the acquisition of program construction skills. My research ex-

amines how to personalize the generation of examples and how to determine the best sequence of examples and problems, based on the student's evolving level of knowledge. It also includes a series of studies created to assess the effectiveness of the proposed technologies and, more broadly, to investigate the role of worked examples in the process of acquiring programming skills.

Results of our studies show the positive impact that examples have on student engagement, problem-solving, and learning. Adaptive technologies were also found to be beneficial: The adaptive generation of examples had a positive impact on learning and problem-solving performance. The adaptive sequencing of examples and problems engaged students more persistently in activities, resulting in some positive effects on learning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

1.1 MOTIVATION

In recent years, the importance of computer science as a component of STEM education has been significantly increasing. In particular, the importance of learning how to program has become broadly recognized and the number of college students taking programming courses has grown rapidly. Programming is considered an important skill for high school students aiming for a career in computer science, but has now grown broadly important to even non-computer science majors as well as to younger, middle school students. This emphasizes the need for developing instructional materials and computer tools that can support students in learning programming. In this context, learning technologies play an important role in increasing the effectiveness of programming instruction and can offer crucial support to students with lower levels of prior preparation, which often includes students from underserved populations. We argue that researchers in the field of cyberlearning should embrace the established best practices in the field of computer science education to develop learning technologies to support and promote these practices.

This dissertation focuses on one of these best practices – *worked programming examples*. 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 text-

book 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, in order to encourage students to explore, run, and modify the examples. Finally, 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].

Therefore, it is surprising that learning technologies for computer science education pay little attention to code examples. In this aspect, computer science education significantly lags behind other STEM areas such as mathematics and physics, where the role of problem-solving examples is comparable to the role of code examples in learning programming. In the field of math and science learning, worked problem-solving examples are extensively studied, particularly in physics [Chi et al., 1989; Conati and Vanlehn, 2000], algebra [Anthony, 2008; Kalyuga and Sweller, 2005; Sweller and Cooper, 1985], geometry [Salden et al., 2009; Schwonke et al., 2007, 2009], chemistry [McLaren et al., 2008], and SQL [Najar et al., 2016].

Despite the importance of examples in the domain of programming, existing research on program examples is limited. Existing program examples focus on either demonstrating program behavior (i.e., what is happening inside a program or an algorithm when it is executed) or program construction (i.e., they illustrate how to construct a program that achieves a specific purpose). As shown in Figure 1, which categorizes existing work on programming examples, advanced types of worked programming examples have not been fully explored. The majority of work has been focused on non-adaptive and non-engaging examples (e.g., [Brusilovsky et al., 2009]). Even in the more explored category of behavior examples, the engagement and personalization aspects have not been explored in combination. Existing engaging examples are not adaptive (e.g., [Naps et al., 2000]) while personalized examples are not engaging (e.g., [Loboda and Brusilovsky, 2010; Yudelson and Brusilovsky, 2005]). Moreover, program construction examples are still limited to primitive static examples with text-based explanations. While person-

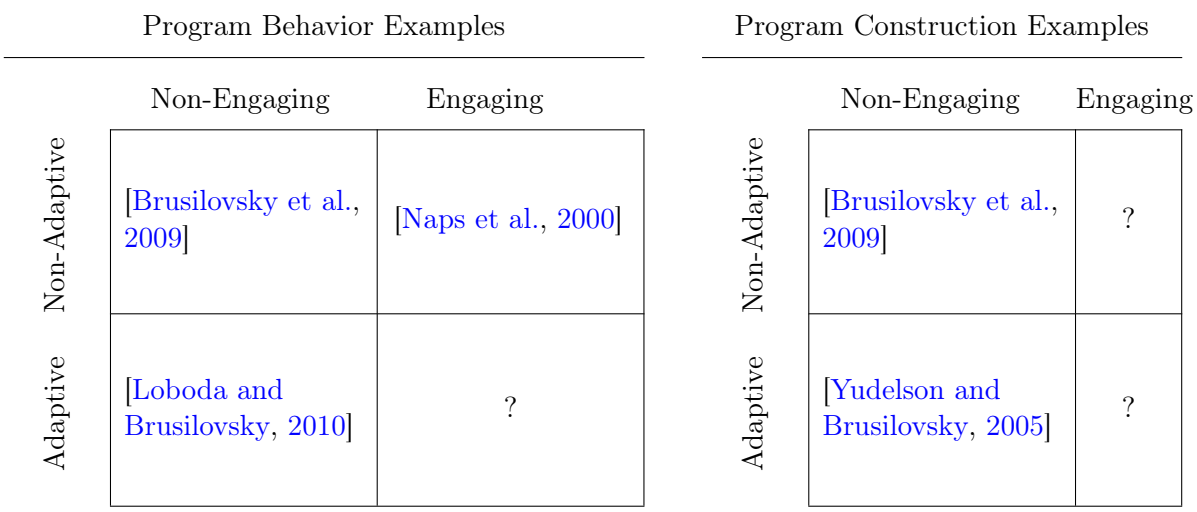


Figure 1: Past work on programming examples categorized by adaptivity and engaging features in examples. Question marks indicate the areas that have not been explored to date.

alization approaches for these examples were explored, no work that aimed to improve engagement in program construction examples has been reported.

In sum, some areas in the research related to worked program examples remain that have not been studied extensively. First, although examples have been consistently proven to be valuable for the student's learning, the field lacks program construction examples with interactive elements that could engage students. Second, research is limited on personalizing access to program examples.

1.2 OVERVIEW AND RESEARCH QUESTIONS

The goal of this dissertation is to investigate the value of the "adaptive" and "engaging" features in programming examples. I limit the scope of my dissertation to program construction examples because, despite the importance of program construction skills,

fewer tools are available for supporting students in acquiring these skills. Furthermore, I focus on the less explored areas within the program construction genre, filling gaps in the existing research (shown in Figure 1 by question marks in the program construction category). Within this context, I am interested in investigating three aspects related to the program construction examples. First, my goal is to design and develop a learning tool for presenting interactive program construction examples, which I refer to as *PCEX*. Next, my goal is to provide support for each student's learning with *PCEX* examples by generating and selecting examples based on each student's current knowledge. Specifically, my goal is to adapt example generation to the student by progressively fading example steps as the student's knowledge increases. Another goal is to personalize the presentation of learning activities during the student's practice to her/his knowledge, in order to guide the student at each stage of learning to the learning activity that could benefit her/him the most in terms of learning programming.

1.2.1 Design and development of interactive program examples (PCEX)

To achieve the first goal of my dissertation, I developed and evaluated an interactive online tool for presenting Program Construction Examples (*PCEX*). I am interested in investigating the following research questions to evaluate the effectiveness of *PCEX* examples on learning how to construct programs, first through an exploratory evaluation of *PCEX* examples (RQ1) and then by comparing the impact of their usage to the use of non-interactive worked examples in a controlled study (RQ2 – RQ4):

RQ1. How much would students use *PCEX* examples on a voluntary basis, and what is the relationship between using *PCEX* examples and student's progress in learning related to programming concepts?

RQ2. Will the *PCEX* examples engage students to work with them more than with non-interactive worked examples?

RQ3. Will working with *PCEX* examples lead to better performance in solving program construction problems than working with non-interactive worked examples?

RQ4. Will working with *PCEX* examples lead to better learning outcomes than working with non-interactive worked examples?

RQ1 was assessed through an exploratory study and by using correlation analysis. RQ4 on the other hand, was assessed through a controlled study, by comparing learning for the control and experimental group.

1.2.2 Adaptive fading in the PCEX examples

To achieve the second goal of my dissertation, I investigated the effect of adaptive fading in the *PCEX* examples on each student's problem-solving performance and learning. The research questions are stated as follows:

RQ5. Would the adaptive fading of *PCEX* example steps, based on a student's current knowledge, lead to better problem-solving performance than by not fading any example steps?

RQ6. Would the adaptive fading of *PCEX* example steps based on a student's knowledge lead to better learning than by not fading any example steps?

1.2.3 Adaptive recommendation of PCEX examples and problems

To achieve the third goal of my dissertation, I wanted to learn how to personalize a student's practice by adaptively selecting *PCEX* examples and problems that match the student's knowledge level at each stage of learning. Therefore, I investigated the effect that the adaptive recommendation of *PCEX* examples and problems had on students engagement in the recommended learning activities as well as on their learning outcomes. The research questions are stated as follows:

RQ7. Would students be more engaged in the *PCEX* examples and problems selected by an adaptive approach compared to a non-adaptive approach?

RQ8. Would the recommendations of *PCEX* examples and problems using an adaptive approach improve a student's learning outcomes more than a non-adaptive approach?

1.3 DISSERTATION TASKS

To answer the research questions in my dissertation, I first developed and evaluated the interactive program construction examples *(PCEX)*[1]. Next, I designed, implemented, and evaluated the personalization technologies to provide personalized access to the *(PCEX)* examples. Thus, I structured my research and development work along the following tasks:

- Task 1: Developing and assessing interactive program construction examples *(PCEX)*.
- Task 2: Developing and assessing personalization technologies for *(PCEX)* examples.

Figure 2 shows how these tasks address the areas where current research on program construction examples is insufficient or non-existent.

Program Construction Examples

	Non-Engaging	Engaging
Non-Adaptive	Already explored	Task 1
Adaptive	Already explored	Task 2

Figure 2: The tasks defined in this dissertation to fill the gaps in the program construction examples.

[1]I received help from a graduate student in my department for developing the current version of PCEX.

In Task 1, *PCEX* examples were developed using an innovative technology that makes examples explorable, meaning that the student can interact with the example and be challenged by tasks that are similar to the example that she/he has viewed. Task 1 was completed by conducting two studies to evaluate the impact of *PCEX* examples on learning programming. In Task 2, personalization technologies were developed to generate and recommend *PCEX* examples. In particular, an adaptive sequencing approach has been implemented to personalize the selection of learning activities in the student's practice sequence ("outer loop" adaptation). Another personalized approach that was implemented was to use adaptive fading, which adapts the amount of support provided within an example to the student's knowledge, by adaptively fading example steps ("inner loop" adaptation). Task 2 was completed by conducting a controlled classroom and a user study to assess the value of the developed technologies on the acquisition of program construction skill.

1.4 CONTRIBUTIONS

This dissertation is the first attempt to systematically explore worked programming examples as a new cyberlearning genre in the domain of computer science education. As part of this dissertation, we developed and evaluated an advanced online learning tool, named *PCEX*, to present interactive worked program construction examples, one of the crucial components of learning how to program. The developed worked examples use an interactive, explorable nature, to improve each student's engagement and learning outcomes. We built personalized technologies on top of this tool to adapt the selection and generation of examples to each student's knowledge, thereby helping students with differing levels of prior preparation.

The findings from the studies in this dissertation gather new insights on how students acquire programming skills from worked examples and how to use examples effectively in computer-supported learning. We hope that our results would lead to the

development of novel learning technologies that actively utilize advanced worked examples in computer science education. Finally, we hope that the developed examples and personalized technologies can be deployed and evaluated in numerous programming courses; thus, directly benefiting larger populations of undergraduate and graduate students.

1.5 THESIS ORGANIZATION

The remainder of this dissertation is organized as follows: Chapter 2 presents the theoretical background related to learning from examples and provides an overview of the role worked examples have played in the field of math and science learning, as well as in learning programming. It also explains the existing research that uses personalization technologies to support each student's work with program examples. The chapter ends with a brief overview of the existing tools for program construction problems as well as some integrated systems that offer practice with both examples and problems.

Chapter 3 describes the design process of interactive Program Construction Examples *(PCEX)*, explains the different aspects of the student's engagement and elaborates on the aspects of those that was focused on in the design of *PCEX*. It also presents the interface, and discusses which interactivity elements were used in the *PCEX* examples to engage students.

Chapter 4 provides an overview of the studies conducted in this dissertation, connecting studies with research questions and hypotheses that were formulated based on the findings of prior research as well as on learning theories. This chapter also presents an overview of the common measures and instruments that were used in our studies. This is done to avoid repeating common details in later chapters.

Chapter 5 and 6 present the evaluation of *PCEX* through two semester-long classroom studies. Chapter 5 describes an initial exploratory evaluation of *PCEX* in an introductory Java programming class (with 71 undergraduate students) which was con-

ducted to understand the student's perception of the tool. Chapter 6 presents a controlled study with a between-subject design that was conducted in a large introductory Python programming class (with 723 undergraduate students), in order to compare *PCEX* with non-interactive worked examples that also focused on program construction.

Chapters 7 and 8 introduce the personalization technologies that we used to provide individualized access to the *PCEX* examples. Chapter 7 explains how the example steps are adaptively faded in the *PCEX* example, based on each student's knowledge. It also reports on the results of a controlled user study with a within-subject design (with 38 participants) that was conducted to compare the adaptive fading of example steps to no fading of example steps. Chapter 8 explains how *PCEX* examples and problems are recommended to each student, using an adaptive recommendation approach. It also presents the results of the controlled classroom study with a between-subject design that was conducted in a large intermediate Java programming class (with 205 undergraduate students), in order to compare the proposed recommendation approach to a non-adaptive approach.

Finally, Chapter 9 summarizes the main findings and conclusions of the dissertation, discusses the limitations of the work, and suggests directions for future research.

2.0 BACKGROUND AND RELATED WORK

This chapter is a literature review of the past research that supports and motivates my dissertation. The work performed by other researchers is described in this chapter with a focus on six areas: (1) worked examples in math and science problem-solving, (2) worked examples in programming, (3) adaptive learning technologies, (4) adaptive technologies for personalizing program examples, (5) program construction assessment tools, and (6) integrated systems for practicing programming. The first four areas provide the theoretical background for my dissertation and review related work on examples. The last two areas review work on systems and tools that are related to the platform that I used in conducting my studies. Specifically, the fifth area reviews the tools for assessing each student's program construction knowledge. The sixth area reviews existing resources that parallel my dissertation, integrating the use of program examples and problems.

2.1 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 such 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 them as models of how to solve certain types of problems. Research on studying worked examples has consistently shown that in 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 positive effect of examples has also been shown in Cognitive Tutor, a particular type of intelligent tutoring system (ITS). Also, Lynnette (for teaching basic equation solving) [Long and Aleven, 2013], Fractions tutor (for teaching conceptual learning of fractions) [Rau et al., 2012], Stoichiometry tutor (for teaching stoichiometry in chemistry) [McLaren et al., 2014], AdaptErrEx (for teaching decimals) [McLaren et al., 2015], Geometry tutor [Salden et al., 2009], Algebra tutor [Anthony, 2008], and Dragoon (for teaching the construction of models of dynamic systems) [Wetzel et al., 2017] are instances of example-tracing tutors that have been shown to enhance learning with an ITS or to make it more efficient by decreasing the instructional time needed.

The examples appear to be most valuable when example presentation is combined with problem-solving. Specifically, past studies found that the pairing of worked examples with practice problems is more effective than providing the learners with only practice problems [Sweller and Cooper, 1985] or examples only [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 "exper-

tise reversal effect" [Kalyuga et al., 2003]. On the 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.

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., 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., 2010]. Secondly, 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]. Thirdly, 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 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.

2.2 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 the work completed for this dissertation. 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 [Brusilovsky and Yudelson, 2008; Esteves and Mendes, 2003; Loboda and Brusilovsky, 2010; Sirkiä, 2013], Javascript [Davidovic et al., 2003], and mini-languages [Brusilovsky, 1994].

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 how a program that achieves various meaningful purposes (e.g., summing an array) is constructed. This distinction might not be clear cut for examples with no use of learning technology, since the same example code could be used for both purposes. However, attempts to augment examples with learning technologies to increase their instructional value (i.e., add code animation or explanations) usually focus on one of these goals.

Program behavior examples have been extensively studied. While 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 more popular. Over the past three decades, a number of specialized educational tools for observing and exploring program execution in a visual form have been built and assessed [Sorva et al., 2013]. Despite their visual and dynamic nature, the majority of tools for presenting animated examples could be considered non-engaging in that they limit the student role to watching the animation passively (e.g., [Miyadera et al.,

2007; Sajaniemi and Kuittinen, 2003; Sirkiä, 2013]). Following several studies that have demonstrated the low effectiveness of "passively-watched" animation 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; Myller, 2006; Naps et al., 2000]. Several studies reported that engaging the students to be more *active* in watching the visualization improved their learning [Byrne et al., 1999, 1996; Hundhausen et al., 2002; 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 a text with comments [Davidovic et al., 2003; Linn and Clancey, 1992; Morrison et al., 2016]. More recently, this technology has been enhanced 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. Research on making a screencast engaging has only recently begun to receive attention, as in [Khandwala and Guo, 2018; Park et al., 2018] which have attempted to make the screencasts more engaging by allowing inline code editing or embedding programming exercises into the videos. An earlier attempt to add interactivity and exploration to worked program construction examples 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]. However, even this approach lacks the engagement power now available to modern technologies for presenting program behavior examples. In this dissertation, 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.

2.3 ADAPTIVE LEARNING TECHNOLOGIES

Another gap in research on worked program examples is personalized access to examples. Personalization is important to address the needs of students with different levels of preparation, as well as being important in the transition to mastery, mentioned earlier in Section 2.1. Thus, a key goal of adaptivity in learning environments is to improve methods for assessing prior knowledge and knowledge growth and then adapting the instruction accordingly [Aleven et al., 2016]. We distinguish two levels for personalizing access to learning contents, namely the "outer loop" and "inner loop" adaptations. These personalization levels have been defined by Vanlehn [2006], to discriminate categories of guidance needed in intelligent tutoring systems (ITSs). The **"outer loop" adaptation** (also called task-loop adaptation [Aleven et al., 2016]) pertains to the selection of an appropriate learning activity for the student while the **"inner loop" adaptation** (also called adaptive scaffolding, or step-loop adaptation [Aleven et al., 2016]) is concerned with the steps taken within the learning activity and the amount of support that the system provides to the student within that activity.

There is good evidence from past studies that adaptive forms of task and step selection, based on the dynamic assessment of each student's evolving knowledge, can substantially contribute to the effectiveness of instruction [Aleven et al., 2016]. For example, adaptively fading support has been found to be effective when the amount of support provided is faded according to a student's needs by including gaps (i.e., faded steps) in the example that the student has to solve [Najar et al., 2016; Salden et al., 2009], when feedback is adapted to the student solution [Mitrovic et al., 2013], or when the amount of support for self-explanation of worked-example steps is adaptively determined, based on a student's knowledge level [Conati and Vanlehn, 2000].

Adapting task selection to each student's increase in knowledge was also found to be effective. Just to name a few research studies that have attempted to do this:

- When in a fixed sequence of learning activities, the type of task (a worked example, a faded example or a problem to be solved) next presented was determined by the amount of assistance the student needed in the previous problems that he/she attempted [Najar et al., 2016]
- When the best example for each problem to be solved was adaptively selected by taking into account 1) the student's current knowledge and 2) the similarity between the candidate example and the problem at hand, in order to promote learning by analogy [Muldner and Conati, 2007]
- When cognitive efficiency (calculated using performance and self-reported values of cognitive load) was used to provide appropriate learning tasks [Kalyuga and Sweller, 2005], or
- When students were guided to program examples using adaptive navigation support techniques (described in detail in Section 2.4.1).

While adapting instructional task and step selection to a student's knowledge growth was proven to be effective across several studies, in some studies (e.g., Kalyuga and Sweller [2005]'s outer loop adaptation and Conati and Vanlehn [2000]'s inner loop adaptation), the positive effective was found only for students with low prior knowledge — that is, as students became more knowledgeable, they needed less structured, adaptive help and the more likely scaffolding would interfere with their work.

In the programming domain, there have been a few recent attempts at providing adaptation to support each student's work with appropriate program examples, a review of which is provided in Section 2.4. Many of the recent attempts to support students in programming aimed at providing inner loop adaptations to the student's problem-solving tasks. One example is the work by Price et al. [2017], which provides on-demand, data-driven hints for each student's programs in iSnap, an extension of Snap!, a block-based programming environment for novices. Other similar attempts were the work

done in [Nguyen et al., 2014; Piech et al., 2015], which presented data-driven approaches to generating hints for students' problem-solving attempts in a programming MOOC and also the work by Rivers and Koedinger [2017], which provided real-time hints to students while they were writing programs in ITAP, a tutoring system for Python programming. Outer loop adaptation has also been explored to support problem-solving tasks. For example, [Hsiao et al., 2010] used a personalized guidance technology known as adaptive navigation support to guide students to questions with an appropriate difficulty level in a Java programming course.

2.4 PERSONALIZING WORK WITH PROGRAM EXAMPLES

There have been very few attempts to personalize student programming instruction with worked examples. Excluding the earlier research of our team (in the Personalized Adaptive Web Systems Lab) on exploring "outer loop" [Brusilovsky et al., 2006; Guerra et al., 2018; Hosseini and Brusilovsky, 2017; Hosseini et al., 2015, 2016; Yudelson and Brusilovsky, 2005] and "inner loop" [Loboda and Brusilovsky, 2010] adaptations for accessing examples, I know of only three other studies that use "outer loop" adaptations for recommending relevant examples when a student fails to solve a problem [Davidovic et al., 2003; Weber and Brusilovsky, 2001; Weber and Mollenberg, 1994]. Table 1 summarizes the related work in terms of their adaptive characteristics by showing the technological (student model and adaptation technology) and educational dimensions (application domain, type of examples, and characteristics of students are used as the source of adaptation). I classified the approaches in Table 1 into three non-exclusive categories, according to the technique by which they provided personalization: personalized access, problem-solving support, and adaptive scaffolding. Figure 3 shows how the related work and this dissertation fit into these classifications.

Table 1: Adaptive approaches for supporting student instruction with problem-solving examples. (Key: ANS: Adaptive Navigation Support; DG: Direct Guidance; PPS: Problem-Solving Support, AP: Adaptive Presentation)

Approach	Domain	Example Type	Student Model
ELM-PE [Weber and Mollenberg, 1994]	LISP	code example	episodic
ELM-ART [Weber and Brusilovsky, 2001]	LISP	solved example	overlay & episodic
SEATS Tutor [Davidovic et al., 2003]	JavaScript	executable code example	overlay
NavEx ADVISE [Brusilovsky et al., 2006], NavEx [Yudelson and Brusilovsky, 2005]	C	annotated example	overlay
WADEIn II [Loboda and Brusilovsky, 2010]	C	program visualization	overlay
Mastery Grids [Hosseini et al., 2015]	Java	annotated example	overlay
Mastery Grids [Hosseini et al., 2016]	Java	annotated example, animated example	overlay
jHelp [Hosseini and Brusilovsky, 2017]	Java	annotated example	overlay
Mastery Grids [Guerra et al., 2018]	Java	annotated example, animated example	overlay

Approach	Adaptation Source	Adaptation Technology
ELM-PE [Weber and Mollenberg, 1994]	knowledge	PSS
ELM-ART [Weber and Brusilovsky, 2001]	knowledge, preferences	PSS
SEATS Tutor [Davidovic et al., 2003]	knowledge	ANS, DG, PSS
NavEx ADVISE [Brusilovsky et al., 2006], NavEx [Yudelson and Brusilovsky, 2005]	knowledge	ANS
WADEIn II [Loboda and Brusilovsky, 2010]	knowledge	AP
Mastery Grids [Hosseini et al., 2015]	knowledge	DG
Mastery Grids [Hosseini et al., 2016]	progress	ANS
jHelp [Hosseini and Brusilovsky, 2017]	knowledge, goal	PSS
Mastery Grids [Guerra et al., 2018]	knowledge	ANS

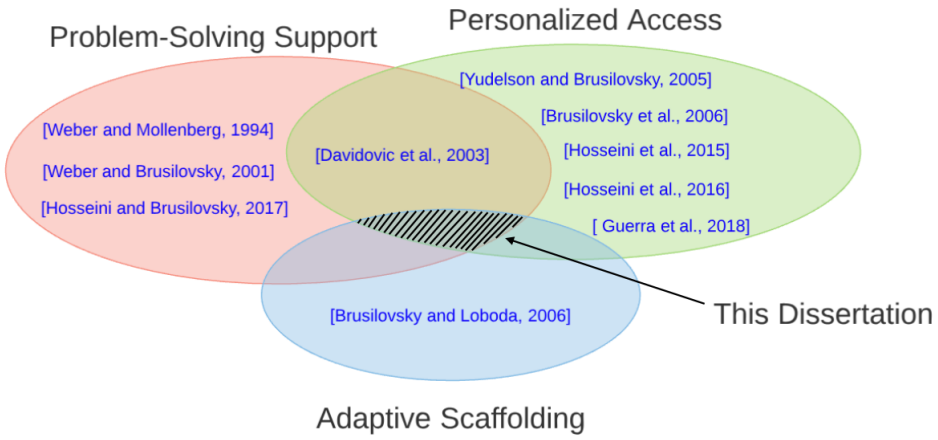


Figure 3: A classification of student modeling and personalization approaches for supporting a student's work with programming examples. The shaded area indicates the contribution of my dissertation to the subareas in adaptive educational systems that support a student's work with programming examples.

2.4.1 Personalized access

Personalized access approaches help students locate examples that match their individual goals, interests, and current knowledge by using combinations of adaptive navigation support technologies such as adaptive annotation [Brusilovsky et al., 2006; Hosseini et al., 2015, 2016; Yudelson and Brusilovsky, 2005] to augment examples, linked with dynamic and personalized visual cues and direct guidance [Hosseini et al., 2015] to provide each student with the most useful example according to her current state of knowledge. All of the systems in [Brusilovsky et al., 2006; Davidovic et al., 2003; Guerra et al., 2018; Hosseini et al., 2015; Yudelson and Brusilovsky, 2005] use concept-based adaptive link annotation to guide students to the most appropriate example in the system.

In both NavEx [Yudelson and Brusilovsky, 2005] and its successor NavEX ADVISE [Brusilovsky et al., 2006], examples that are not-ready-to-be-learned are marked with red cross stop icons, while ready-to-be-learned examples are annotated with fillable

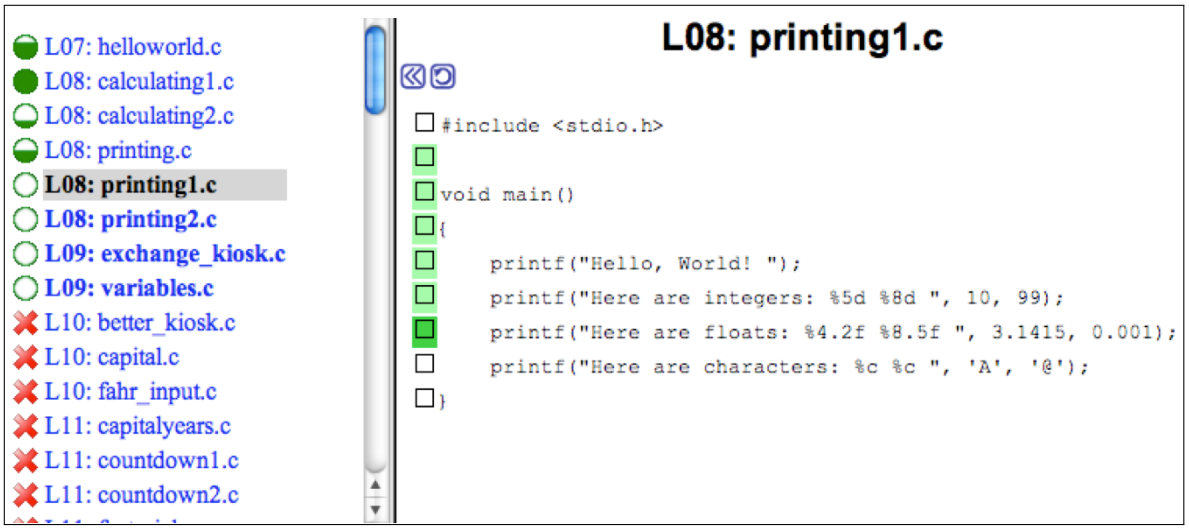


Figure 4: Key to example annotation in NaveEx.

circles. Depending upon the amount of work already completed by a student within an example, the circle would be empty, partially filled, or completely filled in (Figure 4). The SEATS Tutor [Davidovic et al., 2003] uses the same idea for link annotation but implements it in a slightly different way: links to examples that are not-ready-to-be-learned are red, while links to examples that are learned are blue, and links to ready-to-be-learned material are in green.

More recent attempts were presented in Mastery Grids [Loboda et al., 2014], which is a personalized interface designed to help students monitor their topic-by-topic progress and compare it with other students in the class (Figure 5). Mastery Grids provides separate views of course topics (i.e., a row of the grid) to illustrate the progress of the student and her peers in the class. In each row of the grid, each topic cell and all examples that belong to that topic are colored, based on either the student's progress or the progress of the rest of the class, depending on which row the topic cell belongs to. As a result, each student has a personalized view of her progress in course topics and examples that belong to those topics. This information can be used to guide student navigation through the examples and problems presented by Mastery Grids

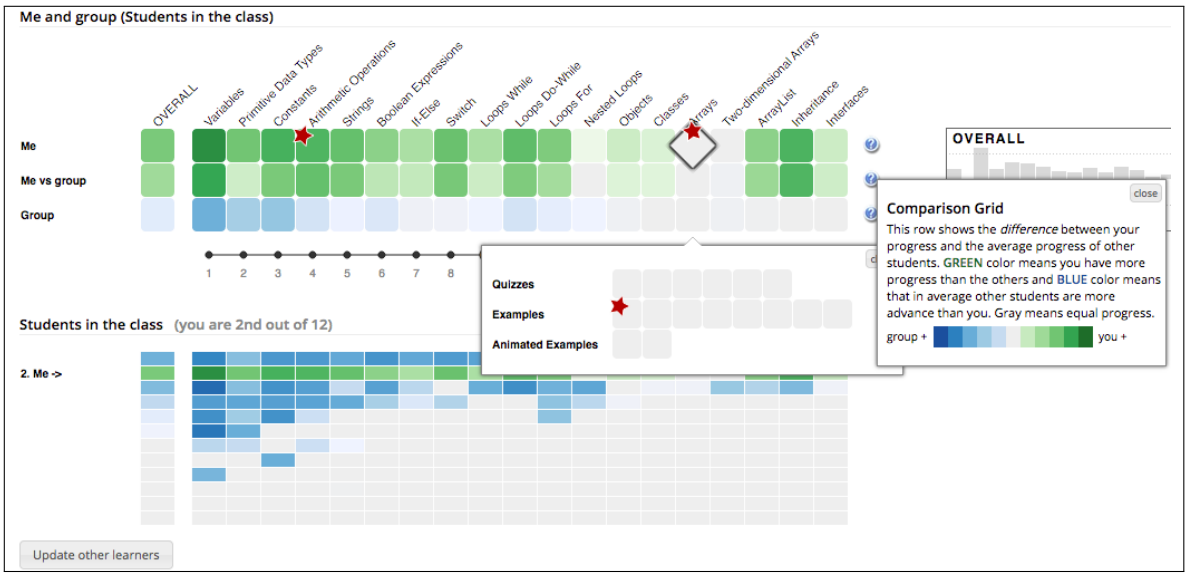


Figure 5: The interface of Mastery Grids with recommendations. The cells with a red star symbol indicate a recommended activity.

[Hosseini et al., 2015, 2016]. In [Hosseini et al., 2015], direct guidance is available in addition to the progress-based coloring of links to examples and problems. The links to examples that the system recommends to be viewed in the next step are marked with a red star (Figure 5). In this study, Mastery Grids uses an adaptive sequencing strategy to maximize the student's gain in knowledge, which recommends the top three (3) examples or problems that the student should explore next. The recommendations are updated after each student attempts to solve a problem. In [Guerra et al., 2018], the topic-level open learned model in Mastery Grids is augmented with a more fine-grained concept-level extension and a learning gauge to guide the student to choose the learning content to maximize learning, either by alerting the student about content that does not provide new knowledge, or alerting the student of content that might be too difficult.

2.4.2 Problem-solving support

Problem-solving approaches, similar to example-based problem-solving support in the ITS domain [Brusilovsky and Peylo, 2003], help students when they are having trouble solving a problem, by providing relevant examples which might be helpful to solve that problem. This approach has been used, for example, in ELM-ART [Weber and Brusilovsky, 2001] and ELM-PE [Weber and Mollenberg, 1994] to suggest relevant successful problem-solving cases to the students from their earlier experience (i.e., examples explained to them or problems solved by them earlier). The SEATS Tutor [Davidovic et al., 2003] provides remedial support by guiding students to an example with the identical or similar structure or pages that help the student in understanding the not-yet-learned prerequisites. Finally, in a recent attempt, we [Hosseini and Brusilovsky, 2017] investigated a range of concept-level similarity approaches that assessed the similarity of problems and examples in terms of programming concepts and by considering student factors, such as expected student's knowledge level and learning goals.

2.4.3 Adaptive scaffolding

Adaptive scaffolding aims to improve learning from worked examples by providing an individualized level of support during the example study. To our knowledge, the only attempt that provided adaptive scaffolding in the domain of programming was presented in WADEIn II [Loboda and Brusilovsky, 2010], which is a system for visualizing expression evaluations in C. WADEIn II adapts animation speed and level of explanation to the level of the student's knowledge. As the student progresses, the speed of these animations increases while explanations are collapsed.

2.5 PROGRAM CONSTRUCTION ASSESSMENT TOOLS

Program construction assessment tools are currently the most popular e-learning technology to help students to acquire program construction knowledge. Early programming assessment tools received uploads of entire student programs and evaluated them against a set of instructor-defined tests [Brusilovsky and Higgins, 2005]. A popular example of these assignment-focused tools is Web-CAT [Edwards and Perez-Quinones, 2008]. More recent tools have evolved into sites which pose questions to students and accept answers in online text editors. Many of these tools ask the students to write small pieces of code, rather than entire programs and insert the student responses into pre-existing starter code. Nick Parlante's CodingBat was one of the earliest examples of these tools [Parlante, 2017]. This model has been adopted by several tools, including CodeWrite [Denny et al., 2011]; CodeAssessor [Zanden et al., 2012]; PCRS [Zingaro et al., 2013]; CloudCoder [Hovemeyer et al., 2013]; and CodeWorkout [Buffardi and Edwards, 2014], which was developed from the Web-CAT project.

All of the aforementioned tools expect the student to write syntactically correct code from scratch, which could take a large and unpredictable amount of time. Therefore, another line of work focused on developing tools that require less time to construct the programs and can be less challenging for students, which is especially appropriate for novices. Parsons and Haden [2006] originally created Parson's problems as an easy way for novices to solve programming assignments without having to type any code or think about the exact syntax of the programming language. The idea is clever: there is a small number of code fragments 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 author introduced a new family of Parson's puzzles inspired by the Python programming language. They proposed a two-dimensional variant of Parson's puzzles 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 the traditional Parson's puzzles. The horizontal dimension is used to define code blocks, based on indentation. In another attempt, Ericson et al. [2017] proposed a 2D Parson's 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.

2.6 INTEGRATED SYSTEMS

While animated examples and program construction assessment tools were originally designed as independent systems, platforms that incorporate more than one type of tool have become increasingly popular in "inverted courses", MOOCs, ebooks [Campbell et al., 2014, 2016; Cooper and Sahami, 2013; Ericson et al., 2015], and online practice systems [Guerra et al., 2018; Hosseini et al., 2016]. As a result, a number of recent tools have been designed to be easily reusable in different contexts. For example, the Online Python Tutor (OPT) [Guo, 2013], which provides memory visualizations for a range of languages, has been incorporated into ebooks [Ericson et al., 2015], MOOCs and online courses [Guo, 2013]. Our work follows this approach. *PCEX* examples were designed as reusable learning content. In the classroom studies conducted in this dissertation, access to examples was provided through an integrated practice system, which also offered coding problems served by the PCRS tool [Zingaro et al., 2013] (Figure 10) as well as Parson's problems [Ihantola and Karavirta, 2011] served by the ACOS server [Sirkiä and Haaranen, 2017] (Figure 17).

3.0 TOOL DESIGN

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

3.1 ITERATIVE DESIGN PROCESS

We followed an iterative design process in the development of *PCEX*. The version we describe in this chapter is the end result of two cycles of design and evaluation. We first designed the mock-ups, then evaluated the mock-ups, then iterated on the design, then developed the working prototypes, then evaluated the prototypes, and finally used the lessons learned to inform the development of a functional interface. To evaluate the mock-ups, we conducted a pilot study and interviewed four students (two males, two females) who were taking an introductory Java programming class at University of Pittsburgh during the Spring 2017 semester. The mock-ups are shown in Appendix A. To evaluate the prototypes, we conducted a usability study with 11 students (eight males, three females) during the summer 2017 semester. Most of the participants were unfamiliar or beginner in computer programming. They were presented with two

examples in Java and two examples in Python and asked various usability questions related to each feature in the prototypes.

3.2 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 intercorrelated and mutually supportive aspects: behavioral engagement, emotional engagement, cognitive engagement, and agentic engagement [Christenson et al., 2012; Reeve, 2013; Reeve and Tseng, 2011].

Behavioral engagement refers to how effortfully involved the student is in the learning activity in terms of attention, effort, and persistence [Skinner et al., 2009]. 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]. 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]. 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 they receive in terms of asking questions, expressing preferences, and letting the teacher know what the student wants and needs [Reeve, 2013].

Engaging examples, in this work, engage students through two of the aforementioned constructs, namely behavioral and cognitive engagement, the aspects most consistently linked with learning outcomes [Bathgate and Schunn, 2017]. That is, we sought to de-

sign examples that would make students be more actively involved in working through the full examples and encouraging them to think more deeply; these forms of engagement would lead to more learning from the examples.

3.3 PCEX: CHARACTERISTICS AND DESIGN

PCEX (Program Construction EXamples) is an interactive tool to support mastering program construction skills through examples. The innovative idea behind *PCEX* is to create "rich examples" that support free exploration and challenge the student. Figure 6 illustrates a *PCEX* example. Each *PCEX* example includes a "goal" (Figure 6, A) and worked program steps (Figure 6, B). The goal states what function the example program performs. The worked steps begin with a subgoal label (Figure 6, C) 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 all or a subset of example lines (Figure 6, D). Once a student clicks on a question mark, an explanation is shown on the right side (Figure 6, E). The student can request additional details for the selected line by clicking on the "Additional Details" button (Figure 6, G) or can navigate to the previous or next line to read an explanation (Figure 6, F).

In addition to being *explorable*, *PCEX* examples *challenge* students by engaging them into a problem-solving activity. When a student clicks on the "Challenge me" button (Figure 6, H), an interactive challenge activity is presented to the student as shown in Figure 7. The goal of a challenge is to encourage students *to apply* the program construction knowledge presented in the original example to *self-assess* whether their understanding is correct. In essence, a challenge is a programming problem that is very

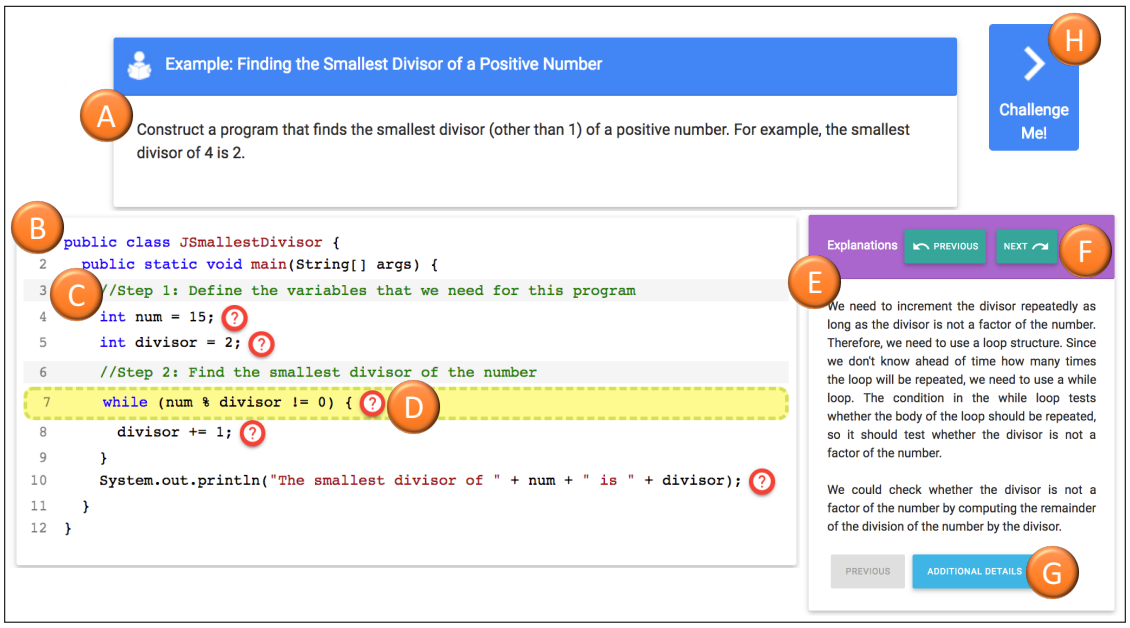


Figure 6: A Java programming worked example in the *PCEX* activity. The example includes the goal (A), interactive worked code (B), the subgoal label presented as a comment (C), the link to instructional explanations (question mark symbols) (D), explanations (E), a navigation link to the explanation for the previous/next line (F), additional details for the highlighted line (G), and a challenge navigation link (H).

similar to the original example in both the goal to achieve and the code. A challenge has a problem statement (Figure 7, I) 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 (Figure 7, J) to each of the missing fields. This drag-and-drop interaction approach is similar to Parsons problems (puzzles) [Parsons and Haden, 2006]. The student can check whether the challenge is solved correctly by clicking on the "Check" button (Figure 7, K). The feedback is presented to the student by highlighting correctly (in green) and incorrectly (in red) placed lines. The student can also request a hint or more detailed feedback (Figure 7, L). If the student cannot solve the challenge with three attempts, she can request the solution.

At any moment, the student can navigate to the core explained example by pressing the "Back" button (Figure 7, M). Also, if an example has several challenges the student may navigate between them by pressing the "Challenge Me" button (Figure 7, N). 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.

In this dissertation, we use the term *PCEX* activity to refer to the *PCEX* worked example and its associated challenges. Each *PCEX* activity was created by annotating the example and challenge code with a set of predefined tags. The annotated code was parsed to generate a corresponding JSON file for each of the *PCEX* activities. The JSON file was used by a single-page JavaScript application to support interactive work with examples and challenges as shown in Figures 6 and 7. The current version of

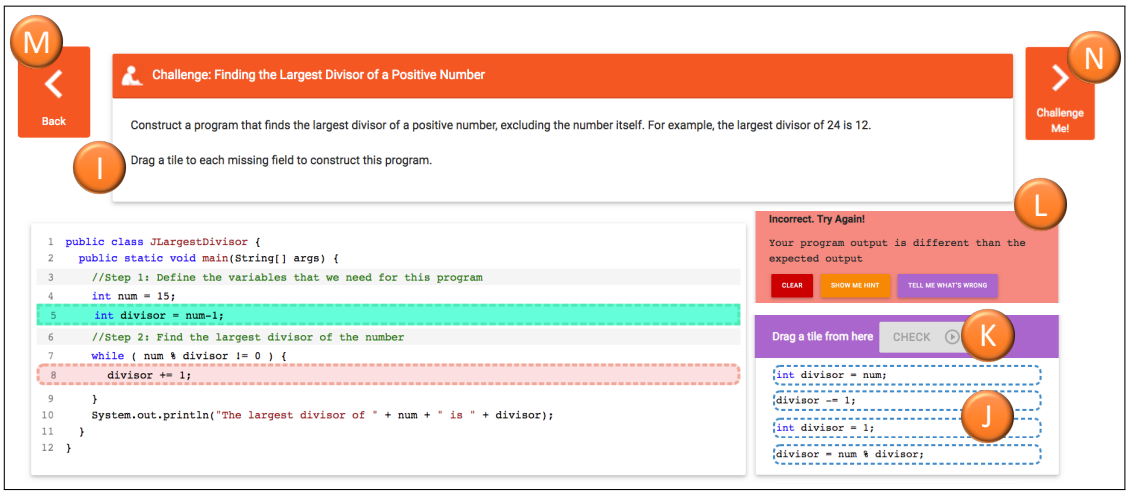


Figure 7: A Java programming challenge in the *PCEX* activity that follows the worked example in Figure 6. The challenge includes the goal (I), has one or more missing lines, and asks the student to drag and drop a line from the given options (J) to each missing line to construct the program. A student can request feedback by pressing the "Check" button (K). The feedback message is shown in part (L). The student can go back to the example by pressing the "Back" button (M). If there are more challenges available, the student can go to the next challenge by pressing the "Challenge Me" button (N). This button is shown only when the current challenge is solved or the student checks the solution after the third incorrect attempt.

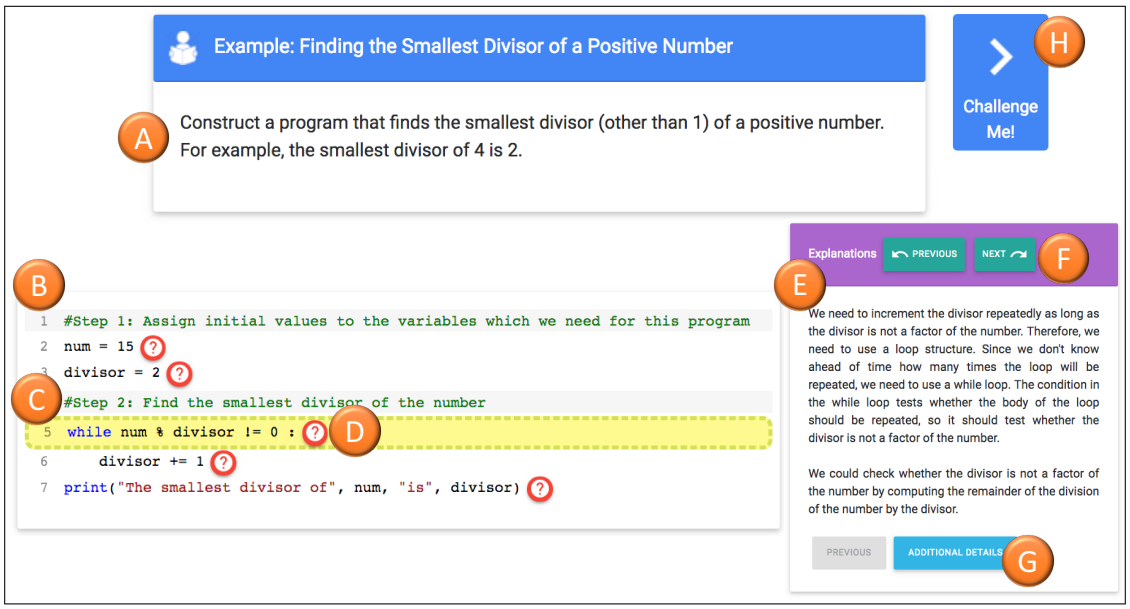


Figure 8: A Python programming worked example in the *PCEX* activity. The example includes the goal (A), interactive worked code (B), the subgoal label presented as a comment (C), the link to instructional explanations (question mark symbols) (D), explanations (E), a navigation link to the explanation for the previous/next line (F), additional details for the highlighted line (G), and a challenge navigation link (H).

the *PCEX* supports any executable code in Java or Python. Note that the program code for a challenge requires to produce an output as *PCEX* employs an output-based evaluation of the student answer.

The Python interface for presenting a *PCEX* activity is similar to the Java interface. Figure 8 illustrates a *PCEX* worked example in Python. As it can be seen from this figure, the interface features are the same as the Java version of the tool. The interface features for presenting the Python challenges are also similar to the Java version of the tool, but with this difference that the Python interface has an additional feature to support indenting the code for the missing lines. The student can decrease/increase the indentation of the line by using the indentation buttons (Figure 9, K/L).

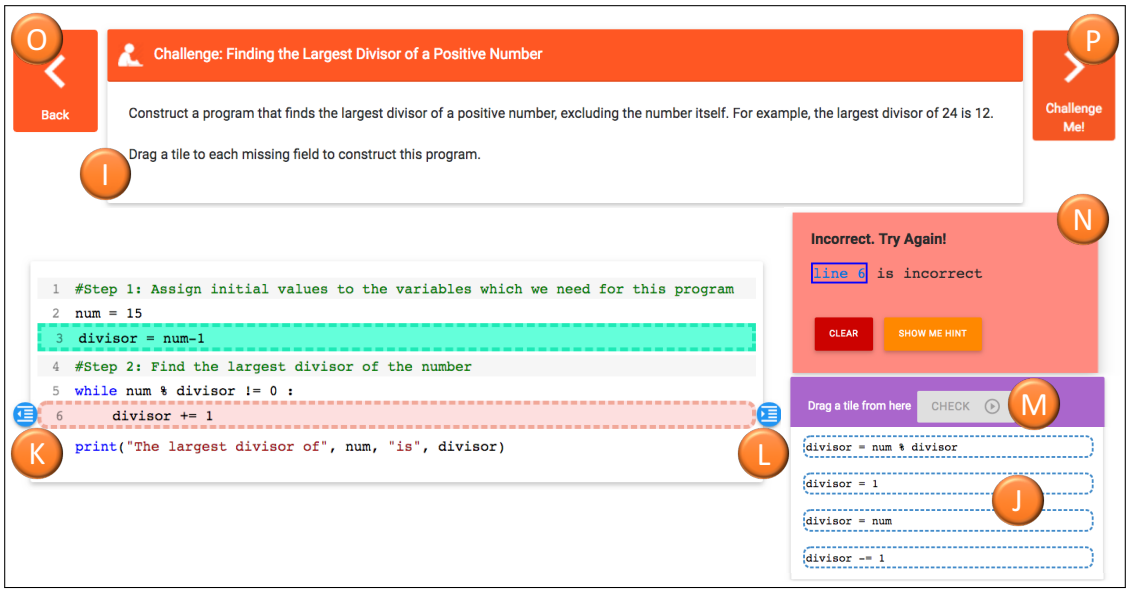


Figure 9: A Python programming challenge in the *PCEX* activity that follows the worked example in Figure 8. The challenge includes the goal (I), has one or more missing lines, and asks the student to drag and drop a line from the given options (J) to each missing line to construct the program. The student 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). 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 the student checks the solution after the third incorrect attempt.

4.0 AN OVERVIEW OF STUDIES

This chapter presents an overview of the studies presented in the following chapters. It also explains how each study contributes to the research questions in this dissertation. Although the studies have different settings, they share some measures and instruments that we used to evaluate our hypotheses, such as pre-test, post-test, system usage variables, and surveys. This chapter provides an overview of these common measures and instruments, in order to avoid repeating the same information for each study.

4.1 CLASSROOM AND USER STUDIES

The goal of the evaluation process was to investigate the value of the "adaptive" and "engaging" features in the proposed program construction examples. We conducted four experimental studies (three classroom studies and one user study) to answer our research questions, which were stated in Section 1.2. Table 2 shows the studies that were employed to answer the research questions, the objective of the studies, and the corresponding chapters describing the studies.

The rest of this section describes the research questions that are investigated in each study. The hypotheses for the research questions were formulated based on the findings of prior studies and the theoretical foundations related to my dissertation work.

Table 2: Relationships between RQs, objectives, studies, and chapters.

Research Question	Objective	Study	Described in
RQ1	Exploratory evaluation of *PCEX* examples	Classroom Study 1	Chapter 5
RQ1, RQ2, RQ3, RQ4	Evaluation of *PCEX* examples relative to non-interactive examples	Classroom Study 2	Chapter 6
RQ5, RQ6	Evaluation of adaptive fading in *PCEX* examples	User Study	Chapter 7
RQ7, RQ8	Evaluation of adaptive recommendation of *PCEX* examples and problems	Classroom Study 3	Chapter 8

4.1.1 Classroom Study 1

The goal of this study was to explore how *PCEX* would be used by students in the classroom and what the relationship would be between working with *PCEX* examples and student's engagement and learning outcomes. The research question addressed in this study is as follows:

RQ1. How much would students use *PCEX* examples on a voluntary basis, and what is the relationship between using *PCEX* examples and student's progress in learning related to programming concepts?

I stated no hypothesis for the first research question as this study was the first deployment of the new tool and there was no past data on which to base our expectations.

4.1.2 Classroom Study 2

This study aimed to examine the impact of *PCEX* examples on student's engagement and learning compared to non-interactive examples. The research questions addressed in this study are as follows:

RQ1. How much would students use *PCEX* examples on a voluntary basis, and what is the relationship between using *PCEX* examples and student's progress in learning related programming concepts?

RQ2. Will the *PCEX* examples engage students to work with them more than with non-interactive worked examples?

RQ3. Will working with *PCEX* examples lead to better performance in solving program construction problems than working with non-interactive worked examples?

RQ4. Will working with *PCEX* examples lead to better learning outcomes than working with non-interactive worked examples?

I hypothesized that the *PCEX* examples would increase student's engagement, problem-solving performance, and learning, compared to non-engaging examples.

4.1.3 User Study

This study investigated the effect of adaptive fading in the *PCEX* examples on student's problem-solving performance and learning relative to not fading any example step. The research questions are stated as follows:

RQ5. Would the adaptive fading of *PCEX* example steps, based on a student's current knowledge, lead to better problem-solving performance than by not fading any example steps?

RQ6. Would the adaptive fading of *PCEX* example steps based on a student's knowledge lead to better learning than by not fading any example steps?

I hypothesized that the adaptive fading of *PCEX* example steps, based on student's knowledge, would lead to better problem-solving performance and learning outcomes than not fading any example steps.

4.1.4 Classroom Study 3

This study investigated the effect of the adaptive recommendation of *PCEX* examples and problems on student's engagement in activities as well as the learning outcomes relative to non-adaptive recommendations. The research questions are stated as follows:

RQ7. Would students be more engaged in the *PCEX* examples and problems selected by an adaptive approach compared to a non-adaptive approach?

RQ8. Would the recommendations of *PCEX* examples and problems using an adaptive approach improve a student's learning outcomes more than a non-adaptive approach?

I hypothesized that adaptive recommendations of *PCEX* examples and problems would lead to more engagement in learning activities as well as better learning outcomes than non-adaptive recommendations.

4.2 OVERVIEW OF COMMON METRICS AND INSTRUMENTS

We used different measures to compare the impact of the *PCEX* examples and the personalization technologies on student's engagement and learning. We grouped these measures into three categories: *engagement*, *performance*, and *learning* metrics. A brief overview of these measures is presented below. More detailed explanations of the measures are provided in later chapters, when each study is presented separately.

4.2.1 Engagement metrics

We aimed to assess behavioral engagement using this category of measures. Specifically, we measured the amount of work on examples and problems, such as by counting the number of attempts on examples, the number of example line clicks to view explanations, the number of attempts on problems, and time on task as an indicator of the

depth of engagement on the attempted activities. In some studies, we also measured the overall usage of the practice system, such as total time spent on the practice system, total attempts on activities, and the number of sessions the student practiced with the system. In Classroom Study 3, we also investigated student's persistence in working with examples and solving problems.

4.2.2 Performance metrics

We aimed to assess problem-solving performance using this category of measures. Specifically, we counted the total number of student attempts and the number of problems that the student solved in the practice system. In Classroom Study 2, we also looked into problem-solving performance on problems that the student solved outside of the system.

4.2.3 Learning metrics

We measured student's learning by using pre- and post-tests. The pre-test and post-test used in each study are presented in Appendix B. In some studies, we also used midterms, final exams, and normalized learning gain as measures of learning. The normalized learning gain is defined as the ratio of the actual gain to the maximum possible gain, based on pre-test performance. Equation 4.1 shows this measure. In this equation, $Posttest_{Max}$ is the maximum possible post-test score.

$$E = \frac{Posttest - Pretest}{Posttest_{Max} - Pretest} \tag{4.1}$$

4.2.4 Survey instruments

4.2.4.1 Example evaluation survey To collect student feedback on the example activities, we administered a two-part survey related to system use and system impact. This survey is presented in Appendix C.1 and it was used in Classroom Study 1, 2, and

3. 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 *bad system experience*, two emphasized *no help needed*, and two addressed *other reasons*, especially, a poor introduction to the system and 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, two negatively and two positively worded. For the learning construct, items referred to student's perception of how much they learned from the examples. For the quality construct, items referred to the quality of the example activities. Finally, for the engagement construct, items examined the level of student involvement in the example activities.

4.2.4.2 Recommendation evaluation survey

To assess student experience with the recommendations in Classroom Study 3, we used a survey, which is presented in Appendix C.2. Similar to the example evaluation survey, described in Section 4.2.4.1, this survey focused only on the students who used the system. This survey included two constructs, namely, *perceived recommendation quality* and *system satisfaction*, which we adapted from the framework introduced by Knijnenburg et al. [2012a,b]. Students were asked to respond using a 5-point Likert scale ranging from *Strongly Disagree* (1) to *Strongly Agree* (5).

5.0 CLASSROOM STUDY 1: EXPLORATORY STUDY OF PCEX

This chapter describes the first evaluation of our new learning tool for presenting interactive Program Construction EXamples (*PCEX*). This study was exploratory in nature and aimed to address RQ1 (described in Section 1.2) to understand how students would use *PCEX* examples and also investigate the relationship between using *PCEX* examples and student's learning of programming.

5.1 RESEARCH QUESTIONS

We have formulated the following research question to build a better understanding of how *PCEX*, the new learning tool that we have developed, would be used by students in the classroom and what would be the relationship between using *PCEX* and learning programming.

RQ1. How much would students use *PCEX* examples on a voluntary basis, and what is the relationship between using *PCEX* examples and student's progress in learning related to programming concepts?

Our research question is exploratory in its nature. We stated no hypothesis as we have no previous data regarding the usage of *PCEX* examples.

5.2 STUDY DESIGN

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

The study followed a pre/post-test experimental design to examine the relationship between usage of *PCEX* and student's learning. At the beginning of the semester, students completed a pre-test, consisting of six questions that evaluated their prior knowledge of the subset of programming constructs covered in the course. The questions were designed to cover both easy and complex concepts in programming. The first three questions asked the student to complete code by filling in blank line(s) or writing small code snippets. The remaining three questions asked the student to determine the correct order of the provided lines of code, in order to achieve a certain purpose. The lines were shuffled and included distractors.

A post-test, isomorphic to the pre-test, was administered at the end of the semester. The maximum possible score on the pre/post-test was 29, 14 points for the first three questions and 15 points for the last three questions. The example evaluation survey (described in Section 4.2.4.1) was also administered at the end of the semester to collect students' perceptions of the *PCEX* examples.

All students were provided with a link and an individual account to access the practice system described in Section 5.3. The practice system included 55 *PCEX* activities (as in Figures 6 and 7) and 46 coding problems served by the PCRS tool (as in Figure 10). The learning content in the practice system was organized into 14 topics. All *PCEX* activities started with a worked example and were followed by one to three challenges (the median was 1). In total, *PCEX* activities included 55 interactive examples with 628 line explanations and 76 challenges. Although the use of the practice system was voluntary, students were encouraged to use the system by offering extra

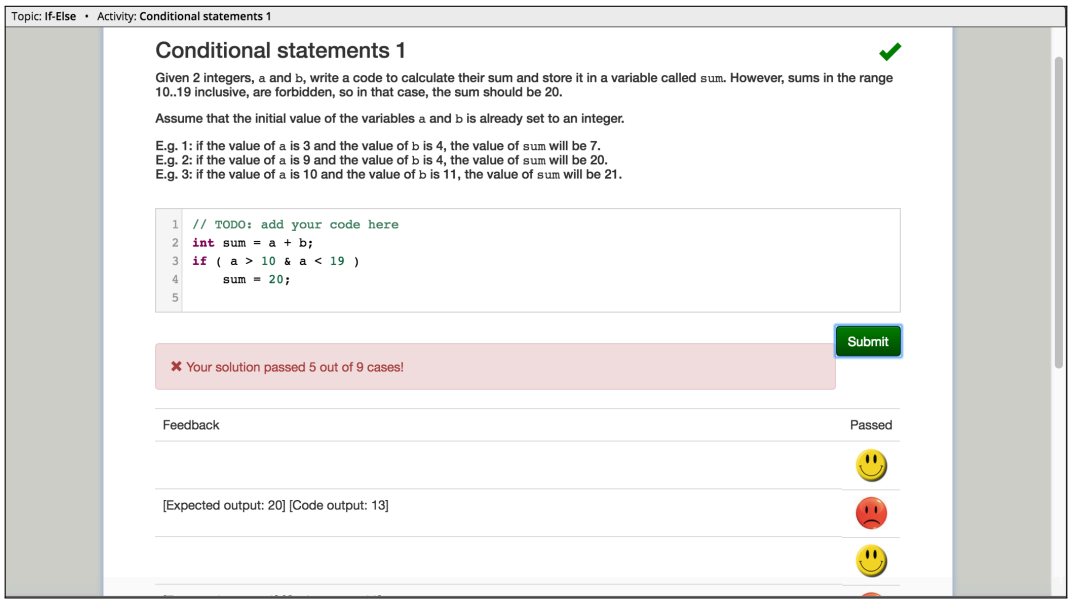


Figure 10: An example of a PCRS problem in our practice system. The feedback message shows whether test cases were passed or not.

credit to those who completed at least three *PCEX* activities (i.e., viewed the examples and solved all the associated challenges for the example they viewed) and solved seven coding problems.

Figure 10 illustrates a PCRS coding problem from our practice system. In this type of problem, the student is asked to write the code for a given task. After submitting the code, the code is tested using a set of unit-tests and the student is shown the result of each test case. If their code fails the tests, they are shown their code output vs. the expected output for the test case. For the subset of failed test cases, the input to the program is also shown. After receiving feedback, the student can modify the code and submit it again without any restriction on the number of attempts.

5.3 PRACTICE SYSTEM

All practice content in our system can be accessed through the Mastery Grids portal [Loboda et al., 2014]. To engage students to work with the content, the Mastery Grids portal provides visual personal progress tracking (known as Open Student Modeling (OSM)), as well as social comparison visualizations (known as Open Social Student Modeling (OSSM)). Only the OSM features of the interface were enabled for this study.

Figure 11 shows the Mastery Grids interface in this study. The grid has one row for overall learner progress and separate rows for each content type. Mastery Grids organizes the content in topics that are represented as a series of colored cells, which get darker as the student completes the content within a topic. The row in the grid shows the current student's topic-by-topic progress by using different shades of green; the darker the color, the higher the progress.

Mastery Grids can integrate different types of smart content. For examples, the smart content in Figure 11 is divided into two types: example-challenges that are *PCEX* examples and PCRS coding problems. By clicking on a topic cell, the student can access content that falls inside that topic. For example, in Figure 11, the student has clicked the topic "While Loops", and the system displays cells to access examples and coding exercises related to this topic.

5.4 COLLECTED DATA

Data from students all sections were combined. 71 students took the final exam. 64 took both the pre-test and post-test and logged in to the system, and 62 attempted at least one activity (i.e., loaded a *PCEX* activity or attempted a coding exercise). Table 3 shows the summary statistics for all usage variables, after removing outliers[1].

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

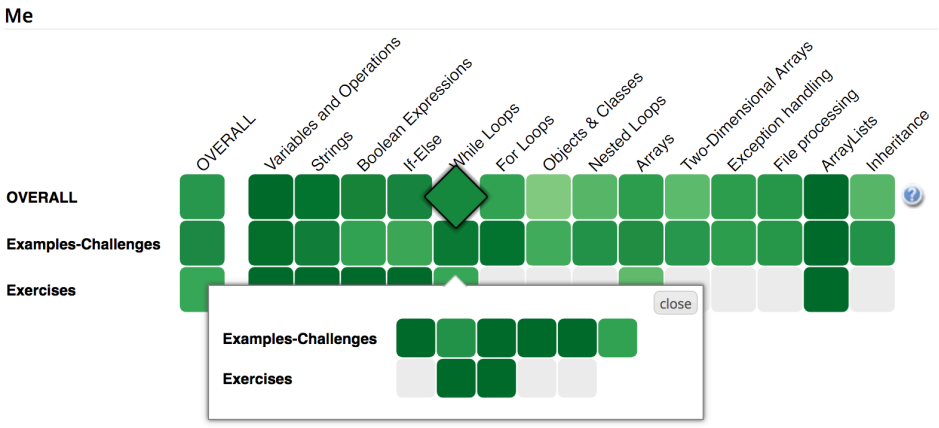


Figure 11: The Mastery Grids interface.

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

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

Table 3: Summary statistics for usage of *PCEX* and coding exercises by students who logged in to the system and attempted at least one activity (N=62).

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

total time that students spent on *PCEX* activities (examples and challenges) was 177 mins., which is comparable to the time spent on coding exercises.

Many of these fine-grained measures were highly correlated with one another and should not be considered as independent measures. In particular, the number of distinct successful attempts on *PCEX* challenges was highly correlated with challenge attempts ($\rho = 0.92$), challenges solved ($\rho = 0.99$), distinct challenge attempts ($\rho = 1$), and time on challenges ($\rho = 0.83$). Similarly, the number of distinct successful attempts on coding exercises was highly correlated with coding exercise attempts ($\rho = 0.88$), coding exercises solved ($\rho = 0.99$), distinct coding exercise attempts ($\rho = 0.99$), and time on

coding exercises ($\rho = 0.9$). *PCEX* activities were also highly correlated to the total time on *PCEX* activities ($\rho = 0.74$). We also found a moderate correlation between example accesses and example line clicks ($\rho = 0.44$) and a strong correlation between example line clicks and time on example lines ($\rho = 0.87$).

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

5.5 RESULTS

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

5.5.1 Relationship between usage of PCEX and student's learning

We evaluated the correlation between usage of *PCEX* activities and student's learning, using several measures of process success and outcomes: (1) learning gain, (2) number of challenges that the student solved; (3) number of coding exercises that the student solved; (4) midterm grade; and (5) final exam grade.

5.5.1.1 Correlation between usage of PCEX and learning gain Learning gain was calculated for the 64 students who had taken both pre-test and post-test, answering all of the questions in the test. The learning gain followed a normal distribution and ranged from 0.07 to 1.0 with a mean of 0.58. The number of example line clicks were not correlated with the learning gain; however, *PCEX* activities completed had

a significant positive correlation with learning gain ($\rho = 0.30, p = .02$). In addition, coding exercises were found to have a strong positive correlation with the learning gain ($\rho = 0.62, p < .001$).

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

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

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

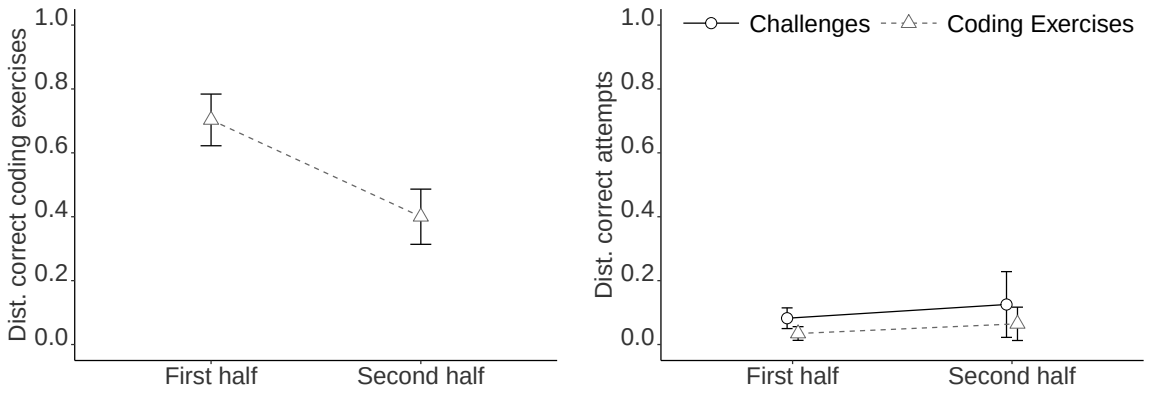
5.5.2 Correlation between usage of PCEX and student's learning over time

5.5.2.1 Correlation analysis during the first and second half of the course

Research studying worked examples has consistently shown that the positive effect of worked examples is stronger in early stages of skill acquisition, when students typically have little or no domain knowledge, while gradually declining in later stages of skill acquisition as the learner develops more expertise [Kalyuga et al., 2003; Sweller et al., 1998]. To investigate whether this relationship exists in our data, we split the data into halves, resulting in data from 55 students in the first half and 44 students in the second half of the course. We fitted regression models to predict the number of distinct coding exercises that student solved in each half using the *PCEX* activities completed and example line clicks. We also fitted regressions to predict the number of distinct challenges that student solved using the example line clicks. In all these regressions, we controlled for differences in pre-test scores.

We plotted the estimated coefficients obtained from the regression analysis in Figure 12. Figure 12(a) shows the estimated coefficients for the *PCEX* activities completed. In both the first and second half of the course, *PCEX* activities completed was significant predictor of the distinct coding exercise solved. More specifically, in the first half, each *PCEX* activity completed was associated with a 0.7 ($SE = 0.1, p < .001$) increase in the number of distinct coding exercise solved. In the second half, each *PCEX* activity completed was associated with only a 0.4 ($SE = 0.1, p < .001$) increase in the number of distinct coding exercises solved (i.e., approximately half the early correlation).

The estimated coefficient for example line clicks was smaller than the coefficient for the *PCEX* activities completed (Figure 12(b)). It was significant only in the first half of the course and only for predicting the number of distinct challenges that a student solved. In the first half, each example line that was clicked increased the distinct correct attempts on challenges by 0.1 ($SE = 0.03, p = .01$). This coefficient was no longer statistically significant in the second half of the course, which suggests that individual line explanations accessible through line clicks are most important in the first half of



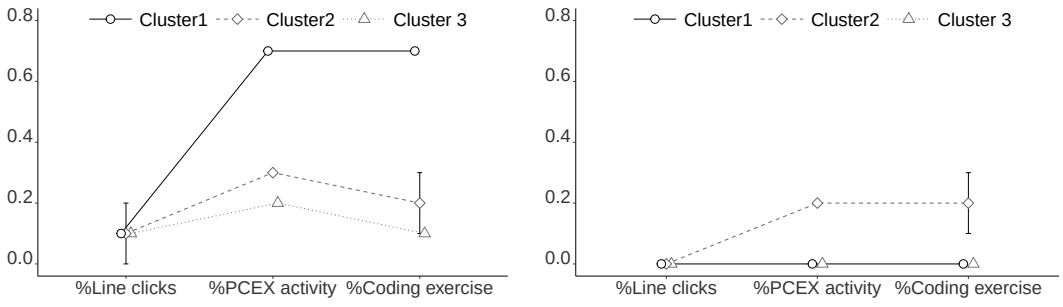
(a) (b)

Figure 12: Regression estimates for (a) *PCEX* activities completed and (b) example line clicks on distinct problems that student solved during the first and second half of the course. Error bars show standard errors. In (b), the solid line represent the estimated coefficients for predicting the distinct challenges solved while the dashed line represents the estimated coefficients for predicting the distinct coding exercises solved.

the course when students are still in the early stages of learning. As students gain more knowledge in the domain, the knowledge added by each individual explanation becomes less essential.

5.5.2.2 Correlation analysis of the regular and exam preparation usage To further investigate how regularity of practice with the system influenced the learning results, we split the total practice of the students into *regular practice* during the semester and *exam preparation practice* (i.e., one week before the exam). Using spectral clustering, we grouped students based on the percentage of example lines clicked, the percentage of *PCEX* activities completed, and percentage of coding exercise solved[2].

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



(a) (b)

Figure 13: Percentage of practice for different clusters when system usage is split into (a) regular and (b) exam preparation. Usage is expressed as mean and standard error for the mean (error bars).

We found three clusters that differed by the activity profile. The amount of practice within each cluster is shown in Figure 13(a) for regular practice and in Figure 13(b) for exam preparation practice.

Students in Cluster 1 had the highest amount of regular practice: On average, they completed 70% of the *PCEX* activities, solved 70% of the coding exercises and clicked on 10% of the example lines. Meanwhile, on average, the students in Cluster 2 clicked the same percentage of lines as students in Cluster 1 but completed 2.3 times fewer *PCEX* activities and solved 3.5 times fewer coding exercises. The students in Cluster 3 had the least amount of regular practice of all the clusters. On average, they clicked on 10% of the example lines, completed only 20% of the *PCEX* activities, and solved only 10% of the coding exercises. As seen in Figure 13(b), Cluster 2 was the only cluster that practiced with the system during the exam preparation week. Students in Cluster 2 had the same amount of practice during the exam preparation week as they had done throughout the semester.

Table 4 summarizes the learning results across different clusters. The learning results were analyzed using a one-way analysis of variance (ANOVA), followed by

Table 4: Summary of learning results for clusters obtained after splitting practice into regular and exam preparation. Values are expressed as mean and standard error for the mean (in parentheses).

Cluster	Pre-test	Learning gain	Exam score	Midterm score
1	4.8 (1.5)	0.7 (0.1)	91.9 (2.2)	94.1 (1.3)
2	5.0 (1.4)	0.5 (0.1)	81.0 (4.6)	83.5 (3.6)
3	4.5 (1.0)	0.5 (0.1)	81.0 (3.9)	80.0 (3.6)

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

5.6 SURVEY ANALYSIS

In the first step of the survey analysis, we assessed the reliability of the survey items under each construct using Cronbach's α. We dropped two items from the engagement construct because item-construct correlations were lower than the recommended value,

0.30. Additionally, we checked whether the internal consistency could improve if any of the items within a construct were deleted. All items in the learning and engagement construct had acceptable internal consistency with the other items within that construct. The α was 0.8 for the learning construct and 0.6 for the engagement construct. After we discarded two items from the quality construct, the α improved from 0.7 to 0.9. No further item was discarded from the survey. In the end, all three constructs appeared to be sufficiently reliable to assess the value of *PCEX* activities, with α values exceeding the suggested minimum acceptable α coefficient of 0.50 [Nunnally and Bernstein, 1978].

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

Distribution of students answers relating to the value of *PCEX* activities is shown in the second plot in Figure 14. The mean learning rating was 3.8 ($SD = 0.8$) which indicates that students perceived *PCEX* activities to be helpful for learning. Notably, more than 70% of the students agreed with the items under the learning construct *(items 1-2, 4-5 in the y-axis)*. The mean quality rating was 3.7 ($SD = 1$), indicating that the students were also positive toward the quality of explanations and code in the *PCEX* activities *(item 3 and item 6 in the y-axis)*. The mean engagement rating was 3.2 ($SD = .9$), very close to the neutral part of the scale. While approximately 40% of the students agreed that they tried hard to understand the examples and challenges and did not skim-read them, a sizable fraction of the class disagreed with these statements *(last two items in the y-axis)*.

Figure 14: The distribution of answers for the survey items. The percentage of respondents who agree/disagree with each item is shown to the right/left of the zero line. The percentage of respondents who neither agree nor disagree are split down the middle and are shown in a neutral color. The items in the y-axis are ordered based on the percentage of agreements, with the uppermost/lowermost item having the most/least agreement.

5.7 SUMMARY AND DISCUSSION

To promote learning, the examples in *PCEX* were enriched by worked steps with subgoal labels and explanations. To assess the relationship between usage of this new educational technology and student's learning, the paper also reported results from a semester-long classroom study. In this study, students enrolled in a Java Programming class were encouraged to use a non-mandatory practice system, which included *PCEX* activities as well as automatically-assessed coding exercises.

When analyzing the collected data, we observed that completing *PCEX* activities had a significant correlation with learning gain. Those students who completed more *PCEX* activities learned more than those who completed fewer (or no) *PCEX* activities. We also found that work with *PCEX* activities had a positive correlation with student's performance in coding exercises even when prior knowledge (as measured by the pre-test) was controlled.

Another interesting observation was that the correlation of work with *PCEX* activities and student's learning was stronger in the first half of the course than the second half. This finding is consistent with past studies that showed that the worked example effect is stronger in the early stages of learning and declines as a student's knowledge grows [Kalyuga et al., 2003; Sweller et al., 1998].

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

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

reasons for lower engagement and discuss options to improve the engagement side of *PCEX*.

Bringing together two sources of information (logs and survey), we can also observe that both approaches to augment traditional examples, line-level explanations and challenges, were valuable for the students. As shown in Section 5.5.1, every explored line explanation and every attempted challenge can be associated with an improvement in student's performance. Students' feedback on survey items that separately assessed the educational value of explanations and challenges was also positive in both cases. At the same time, the data hints that among these two types of augmentation, the challenges are more valuable educationally and more appealing to the students. As Section 5.5.1 shows, the positive correlation of one explored line was more than 10 times lower (0.05 vs 0.62) than the positive correlation of one challenge (which typically expected students to move 2 – 4 lines of code). Students' feedback about the value of explanations was also slightly less positive than their feedback about challenges (3.6 vs 3.9). The same trend can also be observed in the usage statistics reported in Section 5.4: the students completed 51% of the PCEX activities, but explored only 11% of the example lines.

6.0 CLASSROOM STUDY 2: CONTROLLED STUDY OF PCEX

This chapter describes the controlled classroom study that I conducted to address RQ2, RQ3, and RQ4 (described in Section 1.2), in order to measure the effect of *PCEX* examples on student's engagement and learning, compared to non-interactive examples. After revisiting the research questions that are addressed in this study, I present the interface that we developed to contrast *PCEX* with the normal approach for example presentation. Then, I present the study and explain the results.

6.1 RESEARCH QUESTIONS

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, that lack interactivity and a challenge component. Specific hypotheses related to the research questions are presented in Section 6.3.1):

RQ1. How much would students use *PCEX* examples on a voluntary basis, and what is the relationship between using *PCEX* examples and student's progress in learning related to programming concepts? **[Hypothesis 2 and 3]**

RQ2. Will the *PCEX* examples engage students to work with them more than with *textbook-style* worked examples? **[Hypothesis 1]**

RQ3. Will working with *PCEX* examples lead to better performance in solving program construction problems than working with *textbook-style* worked examples? **[Hypothesis 2]**

RQ4. Will working with *PCEX* examples lead to better learning outcomes than working with *textbook-style* worked examples? **[Hypothesis 3]**

6.2 CONTROL GROUP INTERFACE

In this section, we introduce the interface that we developed to compare *PCEX* with the normal approach to example presentation. We used this interface to present worked examples to the Control group of this study. This interface presents the examples in a *textbook-style* form using a simple technology that shows examples statically. Figure 15 illustrates a worked example that was presented, using this static interface. Each example includes a "goal" (Figure 15, A) and worked program steps (Figure 15, B). To make the presentation of the worked examples similar to those in programming textbooks, code segments and explanations are interleaved, by default. The student can click on the "Hide Explanations" button (Figure 15, C) to switch to the mode that presents the code with no explanations. Figure 16 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 Explanations" link (Figure 16, E). The student can go to the next similar example by clicking on the "Next Example" link (Figure 15, D). The next example will be presented in the same style as the previous example. The student can navigate between similar examples using the navigation links.

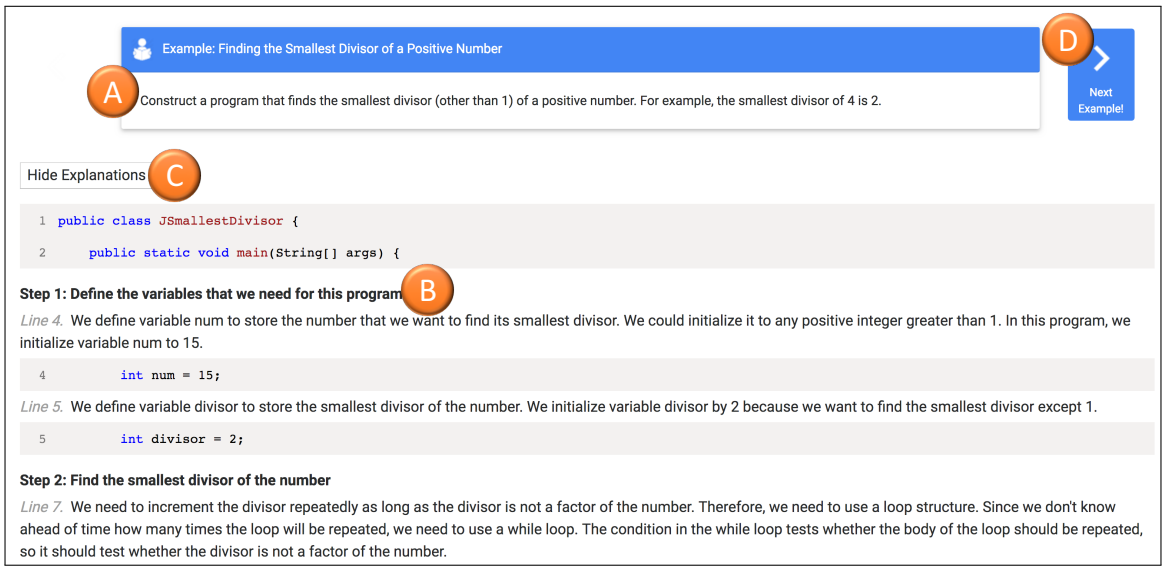


Figure 15: The default mode of a worked example in the Control interface that presents static worked examples for program construction skills. The example is presented in a *textbook-style* form and includes the Goal (A), Worked code with subgoal labels and explanations (B), the link to Hide explanations (C), and a navigation link to the Next similar example (D).When appropriate, a Previous example arrow will appear on the left side.

6.3 THE STUDY

To evaluate the impact on student's engagement and learning from adding explorability and challenges to program construction examples , we conducted a controlled classroom study followed by a survey to collect students' feedback on the usefulness of the new style program construction examples. 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 will conclude this section with information about the objective and subjective measures that were collected in the study and later used in the data analysis.

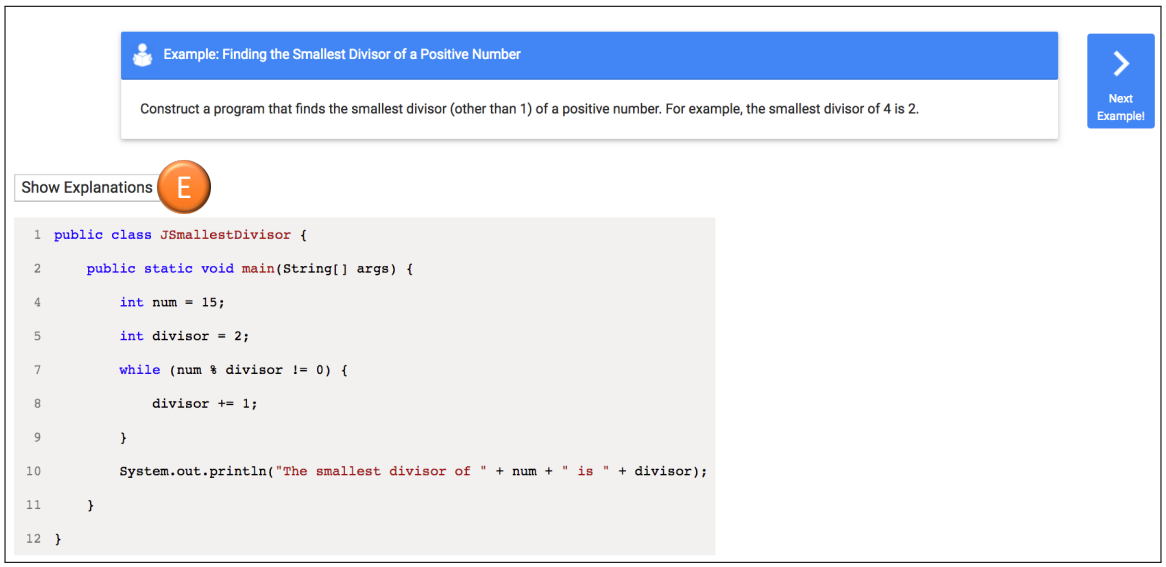


Figure 16: The code-only mode of a worked example in the Control interface that presents worked examples focused on program construction skill. The student can click on link (E) to view the explanations.

6.3.1 Hypotheses

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

*H*1. *PCEX* examples would engage students to work with examples more than with the *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

6.3.2 Study design

To test our hypotheses, we designed a classroom study with two groups. In both groups, students could practice with examples and problems that were accessible through an online practice system (similar to the one described in Section 5.3). 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 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 (Figure 6) and the rest of the examples being presented as tasks that challenged students (Figure 7). In the Control group, by contrast, students could practice with *textbook-style* example activities that presented similar worked examples in the same order, all in the same form as shown in Figure 15.

6.3.3 Study procedure

The study was carried out in an introductory Python programming course. The course was a CS1 service course aimed at students of bachelor programs in various engineering fields at a large university. The students were not Computer Science majors, and most of them complete only one or two programming courses in their bachelor 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 him/herself) 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 two groups. They were given one week to take an online pretest. In the second week of the

class, the practice system (similar to the one described in Section 5.3) 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, but to encourage students to practice with the system, a few extra credit points were given to those who completed eight examples and solved seven problems (similar to Parson's problems, Figure 17). 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 Section 4.2.4.1) and take the online posttest, which was isomorphic to the pretest. Completion of the pre-test, post-test, and survey was voluntary too. Therefore, to increase student 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.

6.3.4 Materials

6.3.4.1 Practice Content Practice content in the system included 52 *PCEX* example activities (as in Figures 8 and 9) and 31 problems similar to 2D Parson's problems (Figure 17). 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. Feedback was shown upon student request, highlighting the correct and incorrect lines in the code.

All practice content was organized into 15 topics that were ordered by increasing difficulty. Each example activity started with a worked example and was followed by one to three challenges (the median was one (1)). In total, the example activities included 52 examples, collectively containing 531 line explanations, and 71 challenges.

6.3.4.2 Pre- and Post-Tests The pretest consisted of 10 questions that evaluated a student's prior knowledge of a subset of programming constructs that were covered

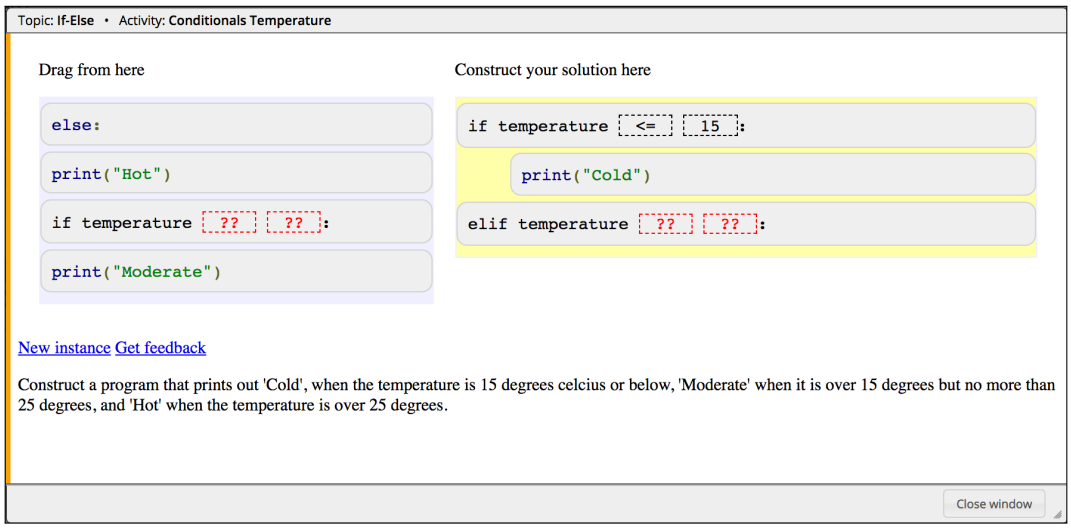


Figure 17: An instance of a Parson's problem in the practice system. The student assembles the solution to the question (written at the bottom) by dragging lines of code to the right side.

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 the students to select the correct code snippet for each given task. The remaining five questions asked the student 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. The maximum possible score on the pre/post-test was 10; one point for each question.

6.3.5 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 5 provides an overview of the metrics we used in our study. As mentioned in Section 4.2, we grouped these into *engagement*, *performance*, and *learning* metrics.

6.3.5.1 Engagement metrics Behavioral engagement metrics focused on 1) work done on the examples and problems, and 2) overall system usage. The specific measures include 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 that were 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 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 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 the 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 the student practiced with the system.

6.3.5.2 Performance metrics We looked into coding performance within and outside of the practice system. The total number of Parson's problems solved and the number of distinct Parson's problems solved were measures of coding performance within the practice system. The measures of coding performance outside of the practice system included: total points that the student earned from coding assignments in the class and how early students submitted their coding assignments.

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.

Table 5: 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 number of Parson's problems solved
		Distinct Parson's problems solved
		Total time on Parson's problems
	Overall system usage	Total practice time
		Total attempts on activities
		Number of practice sessions
Performance	Inside-system problem-solving performance	Total number of Parson's problems solved
		Distinct Parson's problems solved
	Outside-system problem-solving performance	Assignment points
		Early submission
Learning	Near transfer	Pre- and post-test
	Far transfer	Final exam
		• program comprehension questions
		• program construction questions (basic, complex)

Each assignment could be completed during a certain window of time during the term. Sometimes students spent a large amount of time completing the homework and other times they were quicker, apparently better prepared from prior instruction. As a proxy measure for the degree of student preparation, enabling them to complete their homework sooner, we calculated the inverse of the median number of days that homework was submitted after the assignment had been introduced[1].

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

6.3.5.3 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 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 quite simple task, writing a simple class and a main program which 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 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 acceptable reliability.

6.4 RESULTS

6.4.1 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 -0.5), 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 this 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 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 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 number of Parson's problems solved was highly correlated with distinct Parson's problem solved ($\rho = 0.99$); therefore, we excluded the total number of Parson's problems solved from our analysis.

The following subsections present the results from the data analyses. Table 6 summarizes our results by showing the hypotheses of Classroom Study 2 (described in

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

Section 6.3.1), the corresponding data analyses, and whether the hypotheses were confirmed by the data analyses.

Table 6: Summary of hypotheses and results of Classroom Study 2.

Hypotheses	Data Analyses	Measures	Hypotheses Confirmed? Group effect	Hypotheses Confirmed? Interaction effect
H1 – PCEX vs. textbook-style examples: engagement	Section 6.4.2	time-on-task	✓	N/A
		interactions with explanations	x	N/A
H2 – PCEX vs. textbook-style examples: problem-solving	Section 6.4.3	dist. Parson's problems solved	x	x
		assignment points	✓	x
		early submission	✓	✓
H3 – PCEX vs. textbook-style examples: learning	Section 6.4.4	post-test	x	✓
		exam: code comprehension	x	x
		exam: basic code construction	x	x
		exam: complex code construction	✓	x

6.4.2 Engagement analysis

Our first hypothesis ($H1$) was that *PCEX* would make students 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 to compare the group means; for statistically significant cases, effect sizes using eta-squared (η^2) are presented in (Table 7). The metrics in Table 7 are ordered with a decreasing order of effect size. We used Cohen's rules of thumb for interpreting this data, with an effect 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 subtask. The mean of total time spent on the next-to-last subtask was 4 times greater in the Experimental group as compared to the Control group (36.7 vs. 8.3 mins.). The second largest effect was seen for working on

Table 7: 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 Parson's problems solved	14.5 ± 11.9	12.3 ± 11.5	$F(1, 168.82) = 1.56, p = .213$	
Total time on Parson's 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

the first subtask. The total time students spent on first subtasks was about 1.5 times greater in the Experimental group (3.1 mins.) than in the Control group (2.1 mins.).

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, 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 ($M = 4.1, SD = 4.1$) than the Control group ($M = 3.8, SD = 3$). However, both the number of Parson's problems solved and the time spent on Parson's problems were not statistically different between the two groups.

In sum, 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 than the Experimental group students clicked on example lines to view explanations. This difference, however, could be due to how explanations were presented in the *textbook-style* worked examples and *PCEX* examples. As mentioned earlier in Sec-

tion 6.2, 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 conjecture 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 conjecture by showing that, on average, the median of clicks on the "hide explanation" button was twice more than the median of clicks on the "show explanation" button in an example (0.2 vs. 0.1).

6.4.3 Performance analysis

Our second hypothesis ($H2$) was that working with *PCEX* would improve student's performance on program construction tasks more than the *textbook-style* worked examples would. To test hypothesis $H2$, we ran a series of regression analyses–to examine the effect of group, the amount of work on examples, and the interaction between group and the amount of work on examples–for the student 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* – a continuous variable representing the combined work on the first subtasks and next-to-last subtasks in the example activities,
- *Pretest* – a continuous variable representing the student's pretest score, and
- The *Group* × *WOE* interaction – This interaction means that the effect of *Group* on the performance measure is different for different values of work on examples. 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

Table 8: 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 Parson's problems solved, assignment points, and earliness of submission of coding assignments.

	Dist. Parson's 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	−0.02	
Pre-test	.38	.21	.06 .		105.17	40.73	.18*		.05	.01	.29***	
Group × WOE	.02	.02	.07		−2.05	3.43	−0.09		.004	.001	.59***	

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

- *Distinct Parson's problems* that student solved,
- *Total points* that student earned in their coding assignments,
- *Earliness of the student's submissions* in the coding assignments, and
- The *student's exam grade in the program comprehension and program construction questions* of the exam.

The data that was used in these analyses was limited to the 194 students who used the system and 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 8 shows the results of the fitted models.

The results revealed that *WOE* and *Pre-test* were positive predictors of distinct Parson's problems that student solved ($F(4, 189) = 217.2$, $p < .001$, $R^2 = .82$). Importantly, the effect of *WOE* on predicting distinct Parson's problems solved ($\beta = .88$) was about 8 times larger than the effect of 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 Parson's problems correctly, regardless of the treatment group.

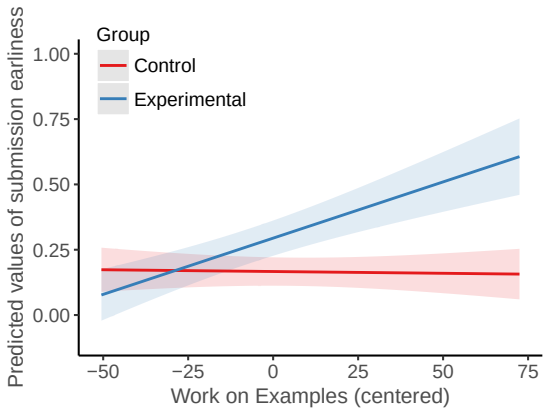


Figure 18: Interaction between work on examples (WOE) and Group factor (Control/Experimental) for predicting the earliness of submissions for coding assignments. Notches indicate 95% confidence interval ranges.

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 to assignment points. Among them, *Group* was the most influential ($\beta = .28$) and *WOE* was the second most influential predictor ($\beta = .26$). We found no significant effect for *Group* $\times$ *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 predictor of earliness of submission overall ($p = .82$), *Group* and *Group* $\times$ *WOE* interaction were a predictor of early submission (Figure 18). 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 ($\beta = .59$), its effect being about two times greater than the effect due to pre-test score ($\beta = 0.29$).

In sum, 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 submission earliness. 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 work with examples and the treatment group on submission earliness was significant, indicating 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 Parson's problems regardless of the treatment group. That is, *textbook-style* worked examples and *PCEX* examples were both helpful for improving student's performance on Parson's problems.

6.4.4 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 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 Section 6.4.3.

Table 9 shows the results of the regression model that tested the effect of *Group*, *WOE*, and *Group* $\times$ *WOE* interaction on the post-test score, controlling for the effect of pre-test score. As expected, *Pre-test* was the most important predictor of student's post-test score ($\beta = .42$). Neither *Group* ($p = .128$) nor *WOE* ($p = .677$) were predictors of post-test score but *Group* $\times$ *WOE* was ($\beta = .29$); suggesting that the effect of group depended on the amount of work on example (Figure 19). As it can be seen

from the figure, work with examples in the Experimental group increased the student's post-test scores more than did work with examples in the Control group.

The results of the fitted models for predicting the exam grade is shown in Table 10. 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 complex question performance. The interaction *Group* $\times$ *WOE* is not statistically significant for any exam category. For the complex program construction questions, *Group* ($\beta = .28$) and *WOE* ($\beta = .23$) are the most and second-most important predictors, respectively, and the effect of each was about two times more than the effect of the *Pre-test* ($\beta = .12$).

In sum, the multiple regression analyses of the learning outcomes supports Hypothesis $H3$, demonstrating a significant positive interaction between work with examples and treatment group on post-test (near transfer test) and a marginal positive effect of treatment group on the complex program construction questions 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 with *PCEX* examples

Table 9: 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*	

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

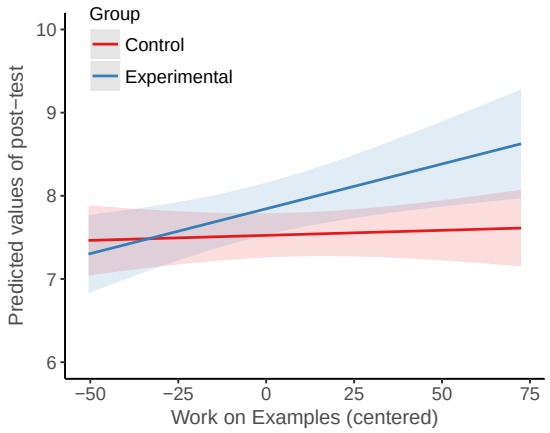


Figure 19: Interaction between work on examples (WOE) and Group factor (Control/Experimental) for predicting the post-test score. Notches indicate 95% confidence interval ranges.

Table 10: 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	−0.66	.75	−0.13	.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		−0.04	.04	−0.15		−0.01	.02	−0.12	

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

was marginally associated with obtaining a higher grade in the complex program construction questions 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.

6.4.5 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 one item from the quality construct because their item-construct correlations were lower than the recommended value, .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 it increased the internal consistency among the items of the construct, improving 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 and Bernstein, 1978].

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 each item in the survey is shown in Table 11. 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 lack of time. Students disagreed with items that suggested other reasons for low/zero 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 varied from 30% to 45%). Overall agreement with each of the items in the engagement construct was low (below 30%).

Table 11: Survey summary in Classroom Study 2.

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.

Thus many students did not perceive examples to be engaging. Figure 20 illustrates the mean ratings in the two study groups for the survey constructs that assessed the value of examples. As it can be seen from Figure 20, in both groups, students were most positive about the quality of examples and the least positive about 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 ($\chi^2(1) = .32, p = .57$).

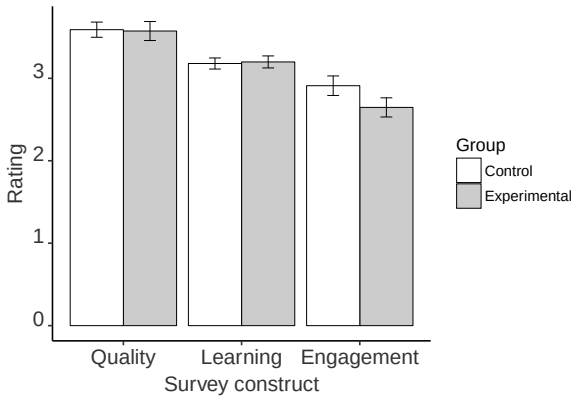


Figure 20: Mean and standard error of group ratings in the survey constructs assessing the value of examples.

The pattern of results did not change when we excluded data of students who used the system less than 10 times or less than 5 times.

6.5 SUMMARY AND DISCUSSION

6.5.1 Summary

This section summarizes and discusses our findings from the classroom study that compared *PCEX* to *textbook-style* worked examples that were focused on program construction skills.

6.5.1.1 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 next-to-last subtasks. The mean of total time on next-to-last subtasks was 4 times more in the Experimental group compared to the Control group. The second largest effect was for the work on the first subtasks. The total time student spent on 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, suggesting that the interactivity element implemented in the first subtask in the Experimental group was not as engaging as next-to-last subtasks (i.e., challenges) were. 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 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.

6.5.1.2 Overall effects on problem-solving performance We found that working on examples and having higher pre-test scores were associated with solving more Parson's problems correctly, regardless of the treatment group. Among these two predictors, the effect of work on examples on predicting distinct Parson's problems solved was about 8 times larger than the effect of pre-test score. We also found that work on examples, group, and pre-test, were positive predictors of assignment points. Notably,

group and work on examples were found to be the most influential influential predictors, with higher predictive power than the pre-test scores were. We also found that the effect of group on submission earliness depended on the amount of work on examples. More work on the examples in the Experimental group is 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 pre-test score.

6.5.1.3 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 student post-test scores more than the work with examples in the Control group. This difference gets larger as the amount of work increases in the Experimental group. Furthermore, work on examples and pre-test were positively predicting 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 group and work on examples was about two times more than the pre-test scores were.

6.5.1.4 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.

6.5.2 Discussion

Our findings from Classroom Study 2 support the positive impact of *PCEX* examples on student's engagement, problem-solving performance, and learning. However, some aspects of our findings need to be discussed further:

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

Secondly, 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 dissertation, are 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.

Thirdly, 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 on task. Similarly, work with *PCEX* examples improved student's learning and coding skills. The difference in the quantitative and qualitative analysis implies that, first, *PCEX* examples engaged students and improved their learning but students did not realize that they were engaged and improving. Second, lower ratings might mean that students in the context of our study had higher expectations for the learning tool. This

is evident when we compare survey responses in Classroom Study 1 and Classroom Study 2. Students in Classroom Study 2 gave lower ratings to all constructs of the example evaluation survey compared to the students in Classroom Study 1. Finally, it might be that students perceived engagement items in the survey differently than how we expected they would.

7.0 USER STUDY: CONTROLLED STUDY OF ADAPTIVE FADING

The chapter describes the controlled user study that I conducted to address RQ5 and RQ6 (described in Section 1.2) to measure the effect of adaptive fading in *PCEX* examples on student's problem-solving performance and learning, relative to the no fading of example steps. This chapter explains the fading strategy, student model that I used to track student's knowledge during the study, study design, and presents the results.

7.1 RESEARCH QUESTIONS

The research questions that will be addressed in this study are stated below. We investigated these questions to build a better understanding of the impact of the adaptive fading in the *PCEX* examples on learning programming relative to not fading any example step. Specific hypotheses related to the research questions are presented in Section 7.4.1.

RQ5. Would the adaptive fading of *PCEX* example steps, based on a student's current knowledge, lead to better problem-solving performance than by not fading any example steps? **[Hypothesis 1 and 2]**

RQ6. Would the adaptive fading of *PCEX* example steps based on a student's knowledge lead to better learning than by not fading any example steps? **[Hypothesis 3]**

7.2 ADAPTIVE FADING STRATEGY

To adapt the example presentation to the student's knowledge, we need to decide "how" and "when" to fade the amount of support in an example. The idea is to help the student rehearse what she/he already knows. Therefore, we fade the amount of support in an example by presenting the example with one or more missing steps that the student has to complete. Any step in the example that the student has acquired sufficient level of knowledge in its concepts are faded. In this study, we used a Bayesian Network, described in Section 7.3, to model the student's knowledge of the programming concepts. The Bayesian Network updated estimates of student's knowledge after the student's attempt on the pre-test problems as well as the problems in Topic 1 and Topic 2.

During the study, when the student wanted to move to the next example in the *Fading* condition, the fading strategy used all of the concepts that appeared in the lines that could possibly be faded, then received the knowledge estimates for those concepts, and then calculated the mastery level for each of those lines by averaging the estimates of concepts inside each of them. If the average of knowledge estimates in a line's concepts was above the threshold, set at 0.7, then that line was faded when the example was presented to the student. In this manner, the adaptive fading strategy adapted to each individual student's evolving level of knowledge, giving the student more practice for what she/he has learned. Note that we set the fading threshold to be 0.7 as it was used by Salden et al. [2009] in a similar study to fade the example steps based on the student's knowledge estimates. Furthermore, this threshold provided reasonable adaptation to student's knowledge when we pilot-tested it on users with different performance levels in the study.

7.3 BAYESIAN NETWORK STUDENT MODEL

To enable personalized access to examples, we first need a student model that could keep track of student's knowledge in the domain concepts. Having such a model enables an effective transition between worked examples and problems, i.e., starting with worked examples, and as the student learns, gradually fading the support in the examples, and finally moving to problems. Various approaches for modeling students have been proposed [Brusilovsky and Millán, 2007]. One of the well-known approaches for modeling a student is the overlay model. An overlay model represents the domain to be learned as a set of knowledge components (KCs) and independently models learner's knowledge of each of these KCs. Bayesian knowledge tracing [Corbett and Anderson, 1995] is a successful example of overlay modeling that has enabled high-quality prediction of student's problem-solving performance and various personalization approaches.

I modeled student's knowledge using overlay models based on Bayesian Networks. The Bayesian Network was developed using the GeNIe modeling environment developed by the Decision Systems Laboratory of the University of Pittsburgh[1]. The network consisted of *concept* nodes that represented the domain knowledge and the *activity* nodes that included those concepts. The concepts were from the Java ontology[2] and were extracted automatically from each activity using a parser [Hosseini and Brusilovsky, 2013] that I developed. To reduce the complexity of the network, the parsed concepts were further reduced to 30% (using TF-IDF) to keep only the most important concepts related to the activity [Huang et al., 2014]. The concepts and activity nodes were indirectly dependent on each other through *Noisy AND gates*. A noisy-AND gate has a high probability of being true only if at least one of its parents is true. The *concept* nodes were parent of the *nosiy AND gates* and the *noisy AND gates* were parent of the *activity* nodes. We used this structure for the network because, in practice, restricting conditional probabilities to noisy-ANDs (or noisy-ORs) significantly reduces

[1] http://dsl.sis.pitt.edu
[2] http://www.sis.pitt.edu/~paws/ont/java.owl

the number of required probabilities and makes the modeling of unobserved variables much simpler [Mayo and Mitrovic, 2001].

The initial estimates for the network nodes were obtained by training the network on the data that was collected from students' usage of the activities in classrooms. In this study, Bayes' theorem was used to recalculate the probability estimates in the network nodes after the student's attempts on the pre-test coding questions. The estimates were also updated after the student's attempts on problems. Every time the student completed a problem, the system updated the probability estimates related to the concepts involved in that problem. The probability estimates increased when the student's answer was correct, and decreased otherwise.

7.4 THE STUDY

7.4.1 Hypotheses

The study tested the following hypotheses regarding the benefits of adaptive *Fading* relative to *No-Fading* of example steps:

- $H1$. Adaptive fading of example steps would improve student's performance on practice problems more than not fading any example step
  - $H1a$. Adaptive fading of example steps would result in solving problems in a fewer number of attempts and in less time than not fading any example step
  - $H1b$. Adaptive fading of example steps would result in investing a lower amount of mental effort on solving problems than not fading any example step
- $H2$. Adaptive fading of example steps would improve student's performance on the test problems more than not fading any example step
  - $H2a$. Adaptive fading of example steps would result in having less problem-solving time in the post-test problems than not fading any example step

H2b. Adaptive fading of example steps would result in higher efficiency in solving the post-test problems than not fading any example step

H2c. Adaptive fading of example steps would result in investing a lower amount of mental effort to solve the post-test problems than not fading any example step

H3. Adaptive fading of example steps would improve student's learning outcomes more than not fading any example step

7.4.2 Study design

The study had a 1×2 factorial design with the within-subject factor "Condition" (*No fading*, *Fading*). Participants practiced two topics (Topic 1 and Topic 2), each under one condition. In the *No fading* condition students practiced with the worked example and problem pairs. In *Fading* condition students practiced with pairs that included a faded example and a problem and the decision to fade an example step was made adaptively based on student's knowledge. The conditions were counterbalanced across subjects to control for order effects from practice or fatigue. Some participants were exposed to the *No fading* in Topic 1, then *Fading* in Topic 2. Other participants received *Fading* in Topic 1, then *No fading* in Topic 2. To reduce the learning effect across the two conditions, we minimized overlap between these two topics. Additionally, to control for the pre-test effects, we balanced the number of low- and high-pretest students when assigning the subjects to each of these condition orders.

7.4.3 Participants and procedure

The study was conducted in Spring 2018 semester and included 38 students from the University of Pittsburgh and Carnegie Mellon University. The requirement for participating in the study was to have little or no familiarity with Java but have knowledge of basic programming concepts in another programming language. The study was conducted in one session lasting about 90 minutes. First, the students read and signed

the consent form for the study and answered a short questionnaire about their prior experience in programming. Then, they were introduced to the system and they were given some minutes to get familiar with worked examples, faded examples, and problems. After that, they took the pre-test (20 minutes). Then, they practiced on Topic 1 and Topic 2 (20 minutes each). In both topics, participants were asked how much effort they invested to complete each problem they attempted. They could respond using a 9-point Likert scale ranging from *very, very low* (1) to *very, very high* (9). In the end, they took the post-test (20 minutes). All participants were compensated $20 upon completion of the study session.

7.4.4 Materials

7.4.4.1 Practice content The study focused on two topics in Java programming. Topic 1 was "Boolean Expression and If-Else" and Topic 2 was "For-Loops and Nested For-Loops". In each topic, there were 8 pairs of isomorphic tasks of increasing complexity. The first task in each pair was an example and the second task in each pair was a problem. Examples were presented by the *PCEX* system (as in Figure 21 for non-faded examples and Figure 22 for faded examples) and problems were presented using the PCRS tool (as in Figure 10).

Figure 21 illustrates a non-faded example in the study. A non-faded example showed the complete code and the link to the explanations for the code lines. The student could click on question mark symbols to view the explanation for the code lines. A faded version of the same example is shown in Figure 22. In the faded example, one or more of the lines were missing (depending on the student's knowledge) and the student had to complete the code by dragging and dropping the given lines into the blank lines. When the student checked the code, the correct (shown in green) and incorrect (shown in red) lines were highlighted. The student could request to view the reason that the answer was incorrect or could ask for a hint. The student could navigate to the problem in the 3rd pair by pressing the button "Next" (located on top of the screen).

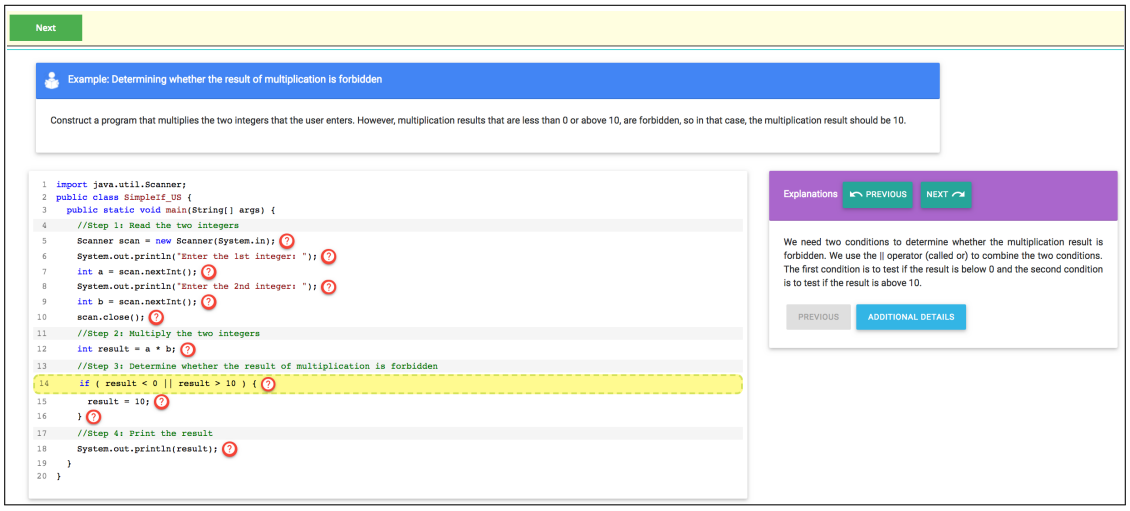


Figure 21: A non-faded example in the 3rd pair of Topic 1 in the *No-fading* condition. The explanation is shown for the highlighted line (in yellow) after the student clicked on the question mark next to that line. At any time, the student could click on the button "Next" to go to the problem in the 3rd pair.

7.4.4.2 Pre- and post-tests

The pre-test and post-test were different but isomorphic tests that consisted of two parts. First, students had to answer 4 questions (2 for each topic) that asked the students to find the correct order of the lines for the given tasks. After that, they had to answer 5 questions (3 for Topic 1, 2 for Topic 2) that asked the students to write the code using the PCRS tool. After each coding question student rated the mental effort as they did during working on the problems in Topic 1 and Topic 2. The time on pre-test was limited to 20 minutes, 5 minutes for the ordering question and 15 minutes for the coding questions.

7.4.5 Metrics

We grouped our metrics into performance and learning metrics, as described in Section 4.2. The performance metrics included: median time on all problems attempted as well as solved problems, the median of perceived mental effort on solved problems, number

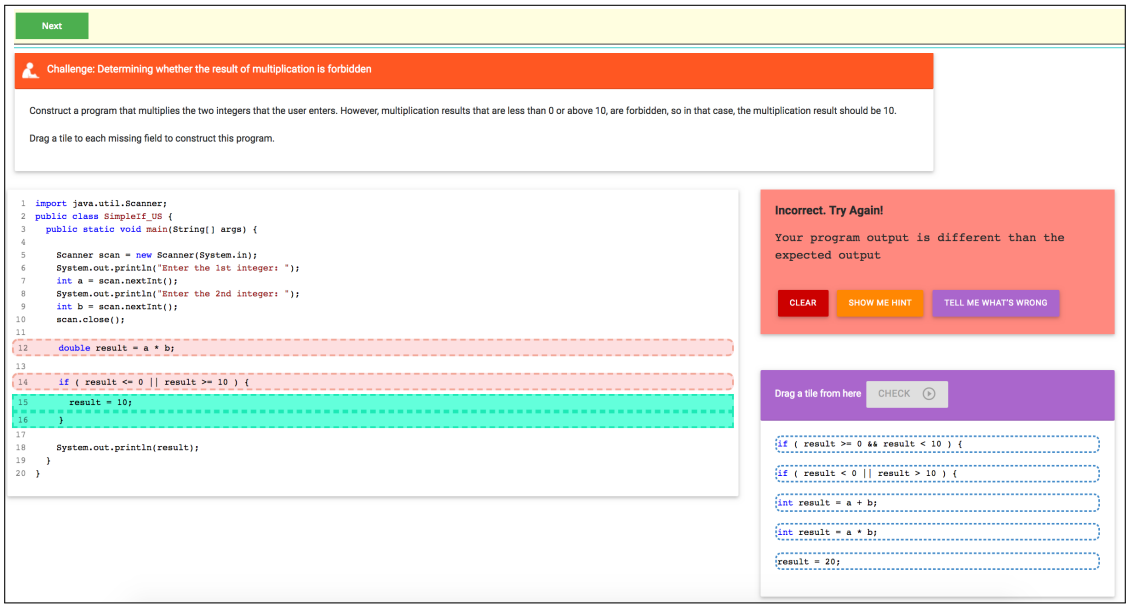


Figure 22: A faded example in the 3rd pair of Topic 1 in the *Fading* condition. The student could click on the "Show Me What's Wrong" button to view the expected output vs. the code output. Also, she/he could request a hint by clicking on the "Show Me Hint" button. The student could click on the "Next" button to go to the problem in the 3rd pair.

of attempts to solve a problem, and efficiency. For the *No fading* and *Fading* condition, these measures were calculated by taking into account the problems that were related to the topic for which students received worked examples and faded examples, respectively. Efficiency was calculated according to the suggestion by Paas and Van Merriënboer [1993], using the z-scores of the median time on problems and median ratio of the tests passed on problems (note that higher values indicate higher efficiency). We used the medians of problem-solving time and mental effort for measuring performance in both practice and test problems. The number of attempts to solve a problem was used as a measure of performance in practice problems and efficiency was used as a measure of test performance.

The learning metrics included pre-test and post-test scores. We distinguished between the overall score and the scores in each of the two parts in the pre-test and post-test (described in 7.4.4.2).

7.5 RESULTS

18 students received *Fading* in Topic 1 and 20 students received *Fading* in Topic 2. One student receiving *Fading* in Topic 1 obtained a very high pre-test score and was therefore discarded from our analyses since there was little to learn. All of the analyses were performed using the data from the remaining 37 students.

The next subsection compares the *No fading* and *Fading* condition in terms of the amount of practice. After that, the results from the data analyses are presented. Table 12 summarizes our results by showing the hypotheses of User Study (described in Section 7.4.1), the corresponding data analyses, and whether the hypotheses were confirmed by the data analyses.

Table 12: Summary of hypotheses and results of User Study

Hypotheses	Data Analyses	Measures	Hypotheses Confirmed? Condition effect	Hypotheses Confirmed? Interaction effect
H1 – adaptive vs. no fading: practice problem-solving	Section 7.5.2	number of attempts	x	✓
		problem-solving time	x	✓
		mental effort	x	x
H2 – adaptive vs. no fading: test problem-solving	Section 7.5.3	problem-solving time	x	N/A
		efficiency	✓	N/A
		mental effort	✓	N/A
H3 – adaptive vs. no fading: learning	Section 7.5.4	post-test	✓	N/A

7.5.1 Overall practice

Table 13 shows the information about example usage in each condition. On average, in the *No fading* condition, students viewed significantly more examples and explanations than in the *Fading* condition but spent significantly less time on examples.

Table 14 shows the information related to the usage of faded examples in the *Fading* condition. The median number of faded steps ranged from 0 to 4 and followed a normal distribution, indicating that the adaptation varied across students (Figure 23). Moreover, the number of faded steps significantly increased ($\rho = .51$, $p < .001$) as students viewed more examples (Figure 24), suggesting that the adaptivity mechanism performed reasonably well in adapting to student's knowledge as it increased the difficulty of the faded examples through time adapting to evolving levels of knowledge. Also, the number of faded examples solved was almost the same as the number of faded examples viewed, suggesting that the students solved almost all of the faded examples that they viewed. Additionally, the median of the number of attempts and requested hints on faded examples that the students viewed was 1. This suggests that the adaptive fading strategy challenged students with an appropriate level of complexity, and thus, was not too aggressive in fading example steps.

Table 15 shows the result of the problem-solving attempts during the practice session. Students solved significantly more problems in the *No Fading* condition than in the *Fading* condition. The number of complex problems attempted and solved was also significantly higher in the *No fading* condition. However, no significant differences were found for the median of problem-solving time between the two conditions.

We also looked into the differences between the two conditions when faded examples were included as problem-solving tasks. When faded examples were included, students solved significantly more problems in the *Fading* condition than in the *No fading* condition. Also, when faded examples were included, there was no significant difference in the number of complex problems solved between the two conditions, even though the number of attempts on complex problems was marginally higher in the *No fading*

Table 13: Descriptive statistics for example usage during the practice session

	No fading					Fading					
	Median	Mean	SD	Min	Max	Median	Mean	SD	Min	Max	Wilcoxon test
TOTAL EXAMPLES											
Non-faded examples	6	5.9	1.4	3	8	1	0.9	1.2	0	6	$V = 666, p < .001^{***}$
Faded examples	–	–	–	–	–	4	3.9	1.9	0	8	–
Non-faded & faded examples	6	5.9	1.4	3	8	5	4.9	1.3	2	8	$V = 370, p < .001^{***}$
MEDIAN EXPLANATIONS VIEWED											
Non-faded examples	1	2.5	3.9	0	18	1	1	0	1	1	$V = 132.5, p = .132$
Faded examples	–	–	–	–	–	0	0.9	2.1	0	7.5	–
Non-faded & faded examples	1	2.5	3.9	0	18	0	0.9	1.9	0	7.5	$V = 287.5, p = .005^{**}$
MEDIAN TIME ON TASK (SEC)											
Non-faded example	33.5	36.5	24.9	10.5	94	18	28.4	27.2	6.4	108.9	$V = 149, p = .257$
Faded example	–	–	–	–	–	92	95.5	30.4	40	171	–
Non-faded & faded example	33.5	36.5	24.9	10.5	94	86	84.4	29	33	168	$V = 30, p < .001^{***}$

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

Table 14: Descriptive statistics related to faded examples in the *Fading* condition

	Median	Mean	SD	Min	Max
Median of faded steps	3	2.6	1.2	0	4
Total faded examples viewed	4	3.9	1.9	0	8
Total faded examples solved	4	3.8	1.9	0	8
Median of requested hints	0	0.1	0.2	0	1
Median of attempts	1	1.2	0.4	1	2.5

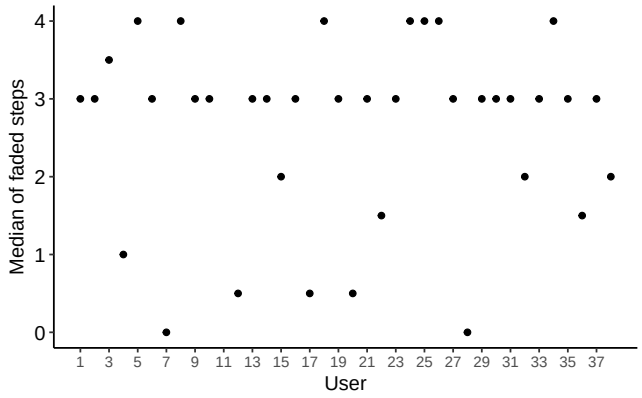


Figure 23: The median of faded steps for each user in the study. Each point represents one user.

condition. And, when faded examples were included, the median of problem-solving time was significantly lower in the *Fading* condition than in the *No fading* condition — which was expected, as faded examples required less time to solve relative to coding problems.

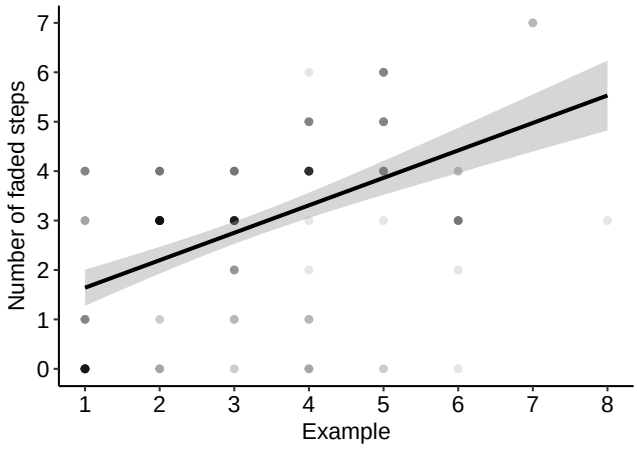


Figure 24: The relationship between the faded examples and number of faded steps. The x-axis shows the order that the examples were presented in the *Fading* condition. The y-axis shows the number of faded steps. Darker color represents higher density of points.

7.5.2 Effects of adaptive fading on performance in practice problems

We first looked into the relative amount of progress during the practice session. All students completed the first two pairs in each condition, more than 70% reached to the 4^{th} pair, less than half reached the 6^{th} pair, and less than 17% completed the last two pairs. Since the number of students who completed the last two pairs was too few, we only focused on the first 6 pairs to compare the impact of the condition on the student's problem-solving performance during the practice session.

Our first hypothesis ($H1$) was that practice in the presence of *Fading* would improve student's performance on practice problems more than *No-fading*. To test hypothesis $H1$, we ran series of mixed models to examine the impact of the condition on the measures related to problem-solving performance, namely the number of attempts to solve a problem, the problem-solving time, and the perceived mental effort on problem-solving. In each model, *Condition*, *Problem*, and interaction of *Condition* with *Problem*

Table 15: Descriptive statistics related to problem-solving attempts during the practice session

	No fading	Fading	
	Mean±SD	Mean±SD	Wilcoxon test
Problems solved excl. faded examples	3.9±2.3	3.2±2	$V = 418.5, p = .036^{*}$
Problems solved incl. faded examples	3.9±2.3	6.9±3.7	$V = 23.5, p < .001^{***}$
Problems solved & partially solved excl. faded examples	4.3±2.2	3.6±1.9	$V = 417, p = .013^{*}$
Problems solved & partially solved incl. faded examples	4.3±2.2	7.4±3.7	$V = 25, p < .001^{***}$
Max complexity level attempted excl. faded examples	2.9±1.3	2.3±.9	$V = 246, p = .004^{**}$
Max complexity level attempted incl. faded examples	2.9±1.3	2.5±1	$V = 193.5, p = .079$.
Max complexity level solved excl. faded examples	2.4±1.6	1.8±1.1	$V = 276, p = .032^{*}$
Max complexity level solved incl. faded examples	2.4±1.6	2.4±1.1	$V = 193.5, p = .832$
Median time on a problem excl. faded examples (sec)	183.2±83.2	180.6±86.5	$V = 383.5, p = .635$
Median time on a problem incl. faded examples (sec)	183.2±83.2	122±53.7	$V = 607, p < .001^{***}$

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

were the fixed effects and participant was modeled using a random effect. The variable *Condition* was a dummy variable representing the condition during the practice with *No fading* condition serving as the reference group factor. The variable *Problem* was a continuous variable representing the problem that student completed in the pair i, $i = 1, 2, ..., 6$.

Table 16 presents the results of mixed model analysis. The results of the mixed model analysis for predicting the number of attempts to solve a problem showed that although *Condition* was not a significant predictor of number of attempts to solve a problem overall ($p = .112$), *Condition* $\times$ *Problem* interaction was (Figure 25(a)). The number of attempts to solve a problem was marginally reduced as the student completed more pairs only in the *Fading* condition.

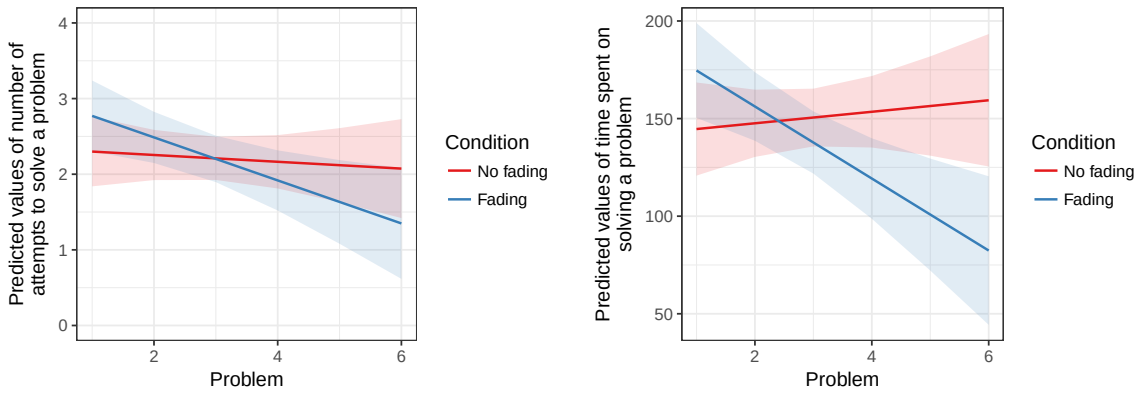
Table 16: Mixed model results of condition, problem, and the interaction of problem with condition predicting metrics measuring performance in practice problems

				Problem-solving time								
	Number of attempts			All problems			Solved problems			Mental effort		
Predictors	β	SE	p	β	SE	p	β	SE	p	β	SE	p
Condition = Fading	.7	.44	.112	29.24	23.59	.216	47.07	21.6	.03*	.19	.32	.56
Problem	−0.02	.09	.813	8.7	4.88	.075 .	6.58	4.67	.16	.07	.07	.33
Condition × Problem	−0.23	.14	.096 .	−15.27	7.21	.035*	−19.3	6.8	.005**	−0.05	.1	.63

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

The interaction between *Condition* and *Problem* was also found to be significant predictor of problem-solving time (columns 2 and 3 of Table 16). Compared to the *No fading* condition, *Fading* significantly reduced the time spent on problems as the students completed more pairs. This relationship existed for all problems as well as problems solved. For the solved problems, *Condition* was also a significant positive predictor of problem-solving time. This suggests that although students in the *Fading* condition spent more time on the solved problems overall, they spent less time on problems as they completed more pairs, even though the complexity of the problems increased (Figure 25(b)). The results of the mixed model analysis did not show any impact of the *Condition* ($p = .56$) or interaction of *Condition* with *Problem* ($p = .63$) on the perceived mental effort for solving problems (column 4 of Table 16).

In sum, the mixed model analyses of problem-solving performance during the practice session supported Hypothesis $H1a$ and $H1b$, demonstrating a significant negative interaction between problem and condition on problem-solving time and a marginally negative interaction between problem and condition on the number of attempts to solve problems. More specifically, more practice and completion of example-problems pairs in the *Fading* condition (and not the *No fading* condition) was associated with solving problems in marginally fewer attempts and also in significantly less time. We found



(a) (b)

Figure 25: Interaction between problem and Condition factor (No fading/Fading) for predicting (a) the number of attempts to solve a problem and (b) time on solved problems. Notches indicate 95% confidence interval ranges. The interaction follows the same pattern for predicting time on all problems.

no differences between *No fading* and *Fading* conditions on the perceived mental effort ratings for the problems students solved.

7.5.3 Effects of adaptive fading on performance in test problems

Our second hypothesis ($H2$) was that practice in the presence of *Fading* would improve student's performance on test problems more than *No-fading*. To test hypothesis $H2$, we first looked into overall condition differences in terms of the mean performance in the post-test problems. Table 17 presents the statistics of the performance metrics in the pre- and post-test. As it can be seen from this table, the *Fading* condition showed improvement on all measures from pre-test to post-test. A similar pattern could be observed for the *No fading* condition on all measures except efficiency. The mean efficiency was lower on post-test than pre-test problems in the *No fading* condition. When looking at the post-test performance, all measures were in favor of the *Fading*

Table 17: Mean and standard deviation of metrics measuring problem-solving performance in test problems

	No fading	Fading
	Mean±SD	Mean±SD
ALL PROBLEMS		
Median time on pre-test (sec)	193.6±81	180.8±67.7
Median time on post-test (sec)	150.5±80	131.8±60.1
SOLVED PROBLEMS		
Median time on pre-test (sec)	153.2±39.4	176±53.3
Median time on post-test (sec)	116.5±60.9	108.8± 40.3
EFFICIENCY		
Pre-test efficiency	−0.02±1.05	.01±.9
Post-test efficiency	−0.21±1.45	.21±1.08
MENTAL EFFORT		
Median of perceived mental effort on pre-test	4.9±.5	4.9±.9
Median of perceived mental effort on post-test	4.0±1.9	3.4±1.7

condition: Compared to the *No fading* condition, the *Fading* condition had, on average, lower median time on problems, higher efficiency, and lower perceived median mental effort ratings.

We used mixed model analyses to examine whether the effect of *Fading* exists when we control for the pre-test performance effect. In all models, *Condition* and pre-test performance were fixed effects and a random effect accounted for the user. The variable *Condition* was a dummy variable representing the condition during the practice, with *No fading* condition serving as the reference group factor. Each variables representing pre-test performance was a continuous variable.

Table 18 presents the results of the fitted models. The mixed model analysis predicting the median time on post-test problems showed that *Median time on pre-test* ($\beta = .32, SE = .1, p = .002$) significantly predicted the median time on all post-test problems but not the solved problem. Although, *Condition* was not a significant predictor of the post-test problem-solving time, it had a negative estimated effect for both all problems and solved problems, pointing to this that Fading tended to reduce the median time spent on a problem compared to the *No fading* condition (the first two DVs in Table 18). Furthermore, results showed that efficiency on post-test problems was significantly related to pre-test efficiency ($\beta = .61, SE = .13, p < .001$) and marginally related to *Fading* ($\beta = .41, SE = .23, p = .09$) (the third DV in Table 18). And, median of mental effort ratings on post-test problems was linearly related to both median of mental effort ratings on pre-test problems ($\beta = .61, SE = .13, p < .001$) and condition ($\beta = .41, SE = .23, p = .089$) (the last DV in Table 18).

In sum, mixed models that we ran to investigate the effect of *Condition* on the student's performance on the post-test problems, supported $H2b$ and $H2c$ but not $H2a$. The mixed model analyses did not show any significant effect of the condition on the problem-solving time, but only a tendency for lower time spent on problems favoring the *Fading* condition. Furthermore, we found that when the effect of pretest problem-solving efficiency was controlled, problem-solving efficiency was marginally higher for post-test problems that were related to the topic that student received faded examples. Finally, when the effect of mental effort ratings at pre-test was controlled, ratings on post-test were found to be significantly lower for solved problems that were related to the topic for which student received faded examples.

Table 18: Mixed model results of predicting performance in post-test problems controlling for the effect of condition and performance in pre-test problems. DVs (dependent variables) are predicted using the IVs (independent variables) shown in that row. The dash symbol (-) indicates that the IV was not used in the model.

	DV											
	Median time on all post-test problems			Median time on solved post-test problems			Post-test efficiency			Median of mental effort on solved post-test problems		
IV	β	SE	p	β	SE	p	β	SE	p	β	SE	p
Condition = Fading	−14.6	12.8	.26	−10.6	11.7	.38	.41	.23	.089 .	−0.53	.25	.041*
Median time on pre-test (sec)	.32	.1	.002**	.02	.12	.85	-	-	-	-	-	-
Pre-test efficiency	-	-	-	-	-	-	.61	.13	< .001***	-	-	-
Median of mental effort on pre-test	-	-	-	-	-	-	-	-	-	.45	.25	.074 .

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

7.5.4 Effects of adaptive fading on learning

Our third hypothesis ($H3$) was that practice in the presence of *Fading* would improve student's learning outcomes. To test hypothesis $H3$, mixed model analyses were performed to check the effect of condition on the overall post-test score and on the coding and ordering questions on their own. However, note that the student score in the ordering questions was found to be moderately correlated with performance on the coding questions (ρ was .4 in the pre-test and .5 in the post-test). In all models, *Condition* and *pre-test score* were fixed effects and a random effect accounted for user. The variable *Condition* was a dummy variable representing the condition during the practice with *No fading* condition serving as the reference group factor. In the model that predicted the total post-test score, pre-test score was the combined score in the pre-test coding and ordering questions. Similarly, in the models predicting the score in ordering/coding questions in the post-test, pre-test score was the score student obtained in pre-test ordering/coding questions.

Table 19 shows the results of the fitted models. The result of the model predicting the total post-test score showed that total post-test score was significantly linearly related to *Condition* ($\beta = .26, SE = .09, p = .008$), and to the total pre-test score ($\beta = .40, SE = .08, p < .001$). The models predicting student score in the post-test coding or ordering questions showed similar results. We found that coding post-test score was significantly linearly related to the *Condition* ($\beta = .15, SE = .07, p = .040$), and to the pre-test coding questions ($\beta = .33, SE = .09, p = .002$). The ordering post-test score was significantly linearly related to the *Condition* ($\beta = .10, SE = .03, p = .005$), and to pre-test ordering questions ($\beta = .35, SE = .06, p < .001$).

In sum, the results of the mixed model analysis supported hypothesis $H3$. We found an overall positive effect in favor of the *Fading* condition on the student's post-test score. Students obtained a higher score in the post-test questions (coding, ordering, and overall) that were related to the topic that they received faded examples.

Table 19: Mixed model results of condition and the pre-test score, predicting the total post-test score, coding post-test score, and ordering post-test score

	DV								
	Total post-test score			Coding post-test score			Ordering post-test score		
Predictors	β	SE	p	β	SE	p	β	SE	p
Condition=Fading	.26	.09	.008**	.15	.07	.04*	.1	.03	.005**
Total pre-test score	.4	.08	< .001***	-	-	-	-	-	-
Coding pre-test score	-	-	-	.33	.09	.002**	-	-	-
Ordering pre-test score	-	-	-	-	-	-	.35	.06	< .001***

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

7.6 SUMMARY AND DISCUSSION

Our findings of the user study supported the positive impact of adaptive fading in terms of several measures related to performance and learning outcomes:

First, adaptive fading influenced practice performance. Two out of three measures that we had for practice performance supported better performance of the student on practice problems. More specifically, more practice in the *Fading* condition (and not the *No fading* condition) was associated with solving problems in marginally fewer attempts and also in significantly less time. On the other hand, we found that students perceived the amount of effort invested on problems to be the same across the practice in *No fading* and *Fading* condition.

Second, adaptive fading influenced test performance. Two out of three measures that we had for test performance supported better performance of the student on test problems. Practice in *Fading* condition resulted in marginally higher problem-solving efficiency and significantly lower perceived mental effort ratings. Moreover, problem-solving time was observed to be lower in the *Fading* condition than the *No fading*

condition but the difference did not reach statistical significance with the number of subjects recruited into the study.

Third, adaptive fading influenced learning. All of the learning measures supported better learning in the *Fading* condition. The post-test score was found to be higher in the *Fading* condition than the *No fading* condition in all cases: in the overall post-test score as well as the scores in each part of the post-test – i.e. ordering questions and coding questions.

One point that should be taken into account when interpreting and generalizing our findings is that in the User Study we compared adaptively faded examples against standard examples to maximize the difference between the two conditions, thus, increasing the chance of registering differences between them. This is a compromise that doesn't allow a reliably measurable separation of the "fading" effects from the "adaptive" effects. Also, there is not a 100% analogy between faded examples in our study and the conventional faded examples in physics or geometry. The faded examples explored in prior studies showed worked steps for all previous steps and the help was faded only for the current step that the student needed to solve. In our study, we presented a faded example by hiding the explanations from the whole code. In that sense, the faded examples in our study changed the code into a mini Parson's problem with no help being shown.

Additionally, the reason that we did not observe any significant difference in the mental effort invested in practice problems may be due to lower reliability of the measurement scale. We argue that it is, generally, harder to register significant differences for the mental effort because it is a subjective measure and students' opinion varies considerably on a rating scale from 1 to 9 such as the one used in our study. As a result, objective measures such as time on task may be less affected by variations in the subject's opinion and may be a better indicator of the student's problem-solving performance.

Another point is the choice of the threshold for fading the example steps in our study. We determined the threshold rather heuristically, first by getting an idea from

similar studies and then by trial-and-error. Even though the study session was short, we observed that the chosen threshold worked reasonably well for providing adaptation to student's knowledge. Yet, we have no evidence that using the same threshold to fade example steps would work for other studies, especially when the student modeling approach is non-probabilistic.

Finally, in the *Fading* condition every subject followed the same sequence of activities. No activity was skipped even if the subject had obtained sufficient knowledge in the concepts of that activity. However, as shown by Najar et al. [2016], the students could learn more and faster when the adaptive strategy allows skipping of examples or problems. Therefore, we speculate that our results could be improved further when the adaptive strategy allows skipping of faded examples or problems for the students who have sufficient knowledge of the concepts of those activities.

8.0 CLASSROOM STUDY 3: CONTROLLED STUDY OF ADAPTIVE RECOMMENDATION

The chapter describes the controlled classroom study that I have conducted to address RQ7 and RQ8 (described in Section 1.2) to measure the effect of the adaptive recommendations of *PCEX* examples and problems on student's engagement on activities and learning, relative to the non-adaptive recommendations. This chapter first explains the adaptive recommendation approach that was used to recommend examples and problems in the student's practice sequence. Then, it explains the study design, and finally presents the results.

8.1 RESEARCH QUESTIONS

The research questions that will be addressed in this study are stated below. We investigated these questions to build a better understanding of the impact of the adaptive recommendation of the *PCEX* examples and problems on learning programming relative to the non-adaptive recommendation. In this study, the non-adaptive recommendations are generated by a random selection of examples and problems within each topic in the course. Specific hypotheses related to the research questions are presented in Section 8.3.1.

RQ7. Would students be more engaged in the *PCEX* examples and problems selected by an adaptive approach compared to a *random* approach? **[Hypothesis 1]**

RQ8. Would the recommendations of *PCEX* examples and problems using an adaptive approach improve a student's learning outcomes more than a *random* approach? **[Hypothesis 2]**

8.2 ADAPTIVE RECOMMENDATION STRATEGY

We have two types of recommendations for guiding the student to the learning materials that match his/her knowledge: *proactive* and *reactive*. Both recommendations provide the "outer loop" adaption by using the student model to determine when to present examples, faded examples, or problems in the student's practice sequence so that the practice sequence matches student's knowledge needs. In *proactive* recommendation, the system proactively suggests learning materials to the student in order to advance his or her knowledge. In the reactive recommendations, the recommendations are generated in reaction to the student's failures in problem-solving. If the student fails in solving a problem, the system recommends related examples to the student to help the student to solve that problem. Our recommendations do not force the student to go to the recommended content and the student is free in making her/his navigational steps, as in our previous work [Hosseini et al., 2015]. The subsections below describe our approach to generating *proactive* and *reactive* recommendations.

In both types of recommendations, all estimates for the student's knowledge are obtained from the Bayesian Network student model described in Section 7.3. The estimates were updated after the student attempts on the pre-test problems, and challenges and coding exercises in the practice system.

8.2.1 Proactive recommendation

The proactive recommendations are generated after the student completes an activity and based on the student's goal. In the practice system, the student determines

his/her goal by selecting a topic. So, when the student clicks on a topic in the practice system, proactive recommendations will select the next best learning material for the student for mastering the topic. Figure 26 illustrates the recommendations within the "If-Else" topic. The recommendations within the topic are shown as a ranked list with the strongest recommendation being the top element in the list. The cell of the recommended activities is marked with a star symbol. The size of the stars represents the importance (rank) of the recommendation. The top recommendation in the list has the largest star. To see the recommendations, the student could click on the cell with star symbols or on the activity names on the recommendation list.

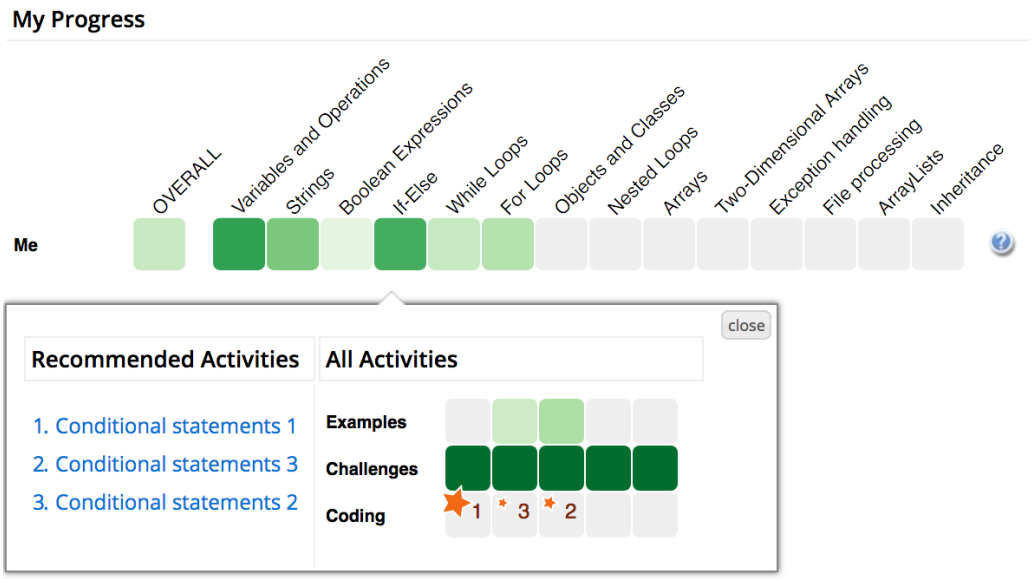


Figure 26: The presentation of recommendations in the practice system interface. The top–3 recommended activities are shown as a list. The cell related to each recommended activity is marked with a star symbol. The size of the stars is relative to the position of the recommended activity in the top–3 list, the first recommended activity has the largest star. To navigate to a recommended activity, the student could click on the activity in the list or on a cell with a star symbol.

My recommendation approach for selecting program examples and problems is grounded in the learning theories, particularly "worked example effect" Sweller et al.

[1998] and "expertise reversal effect" [Kalyuga et al., 2003]. According to these theories, studying worked examples is more beneficial than problem-solving in the early stages of skill acquisition when the learner typically has little or no domain knowledge (worked example effect). However, this advantage disappears over time as the learner develops more content expertise and in that stage problem-solving is superior to studying worked example (expertise reversal effect). My recommendation approach determines the most beneficial activity within the topic that student has selected by examining how prepared the student is to attempt each activity. It aims to select the three most beneficial activities within the topic that the student plans to practice. Depending on the student's knowledge, the most beneficial activities could be examples, or problems (i.e., challenges or coding exercises), or a combination of both. The recommendation approach attempts recommending problems first. If we can find no such problem, it indicates that the student is still not prepared for problem-solving. In other words, it indicates that the student is at an early stage of learning in the topic; and thus, not yet ready to transition to problem-solving. As a result, the recommendation approach attempts to select examples within the topic.

The flowchart in Figure 27 shows the process of generating recommendations for a typical topic in five ordered steps: (1) generating activity ranked lists, that creates separate ranked lists for ordering examples, challenges, and coding exercises based on how ready student is to attempt the activities within each content type ; (2) attempting to select problems, that aims to select the most helpful problems (i.e., coding exercises or challenges), if any; (3) attempting to select examples, that aims to select the most helpful examples, if any; and (4) attempting to select the problems given the student's knowledge. The process stops at any time when the recommendation list is complete, — i.e., three most helpful activities are selected.

Step 1: Generating activity ranked lists

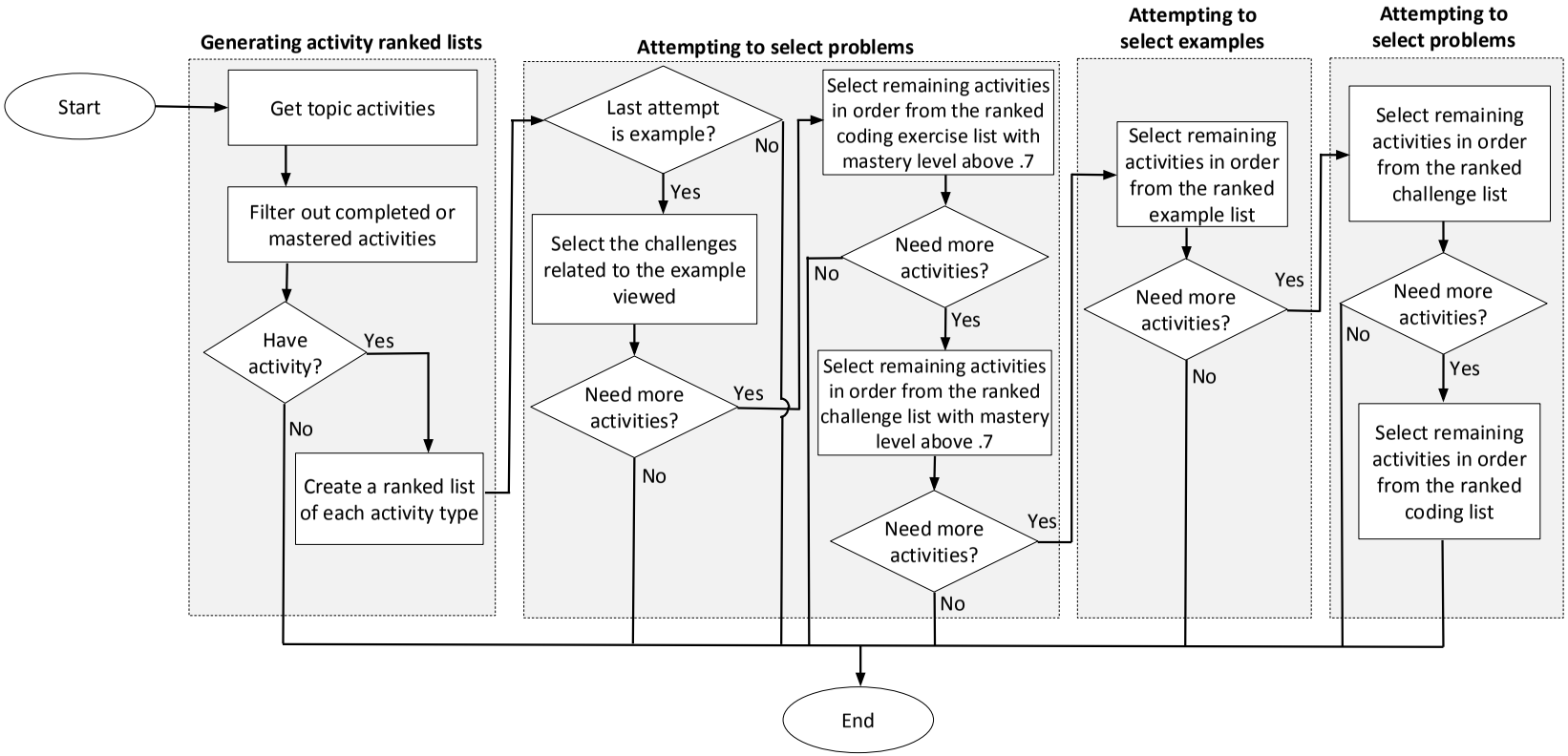


Figure 27: Proactive recommendation flowchart

In this step, the ranked list of activities will be generated. We create one ranked list for each activity type (i.e., examples, challenges, or coding exercises). This is done by first getting the list of activities within each topic. We reduce the list to activities that are not completed, and not mastered. After that, we generate the three ranked lists, one for each activity type. The idea is that the activity that is closest to the student's zone of proximal development is ranked higher on the list. The activities in the ranked list of challenges or coding exercises are ordered based on decreasing values of probability of solving the challenges or coding exercises. The activities in the ranked examples list are ordered based on the probability of understanding the examples which is the mean probability of knowing the example's concepts. In case of ties, the activity that has a fewer number of concepts will be ranked higher on the list.

Note that the Bayesian Network does not take into account the student's work on examples. As a result, it makes no prediction on what is the probability that student could understand an example. I calculated this probability by averaging the probability estimates for each of the concepts in the examples. To determine whether the student could understand the example, one could use the estimate for the weakest concept or, on the other hand, an average of the estimates for all concepts in the example. I chose to use the average because I found it more suitable for representing student's knowledge in a program code. Typically, each program has many concepts and, as a result, understanding the examples depends on how much the student knows each of those concepts. Thus, the average of mastery in all example's concepts is a better indicator of how much the student understand the example as opposed to considering the mastery in a single concept such as the weakest concept.

Step 2: Attempting to select problems

In this step, we search the ranked activity lists for findings problems that the student is ready to attempt. If the previous attempt is an example, we select all challenges in the list of ranked challenges that are related to the example viewed. The idea is

that we challenge the student to practice the concepts in the example. If the previous attempt is not an example or the number of challenges related to the example viewed is less than three, then we start searching first in the ranked list of coding exercises and then in ranked list of challenges to select problems that are within the student's zone of proximal development. We determine whether an activity is within the student's zone of proximal development by comparing the probability of solving that activity to a threshold of 0.7. If the probability of solving the activity is above 0.7, we consider that activity to be within the student's zone of proximal development. We selected this threshold on a trial-and-error basis because during several rounds of pilot-testing it generated reasonably good recommendations. We search the ranked list of challenges only if we could not select all three of the recommended activities from the ranked list of coding exercises.

Step 3: Attempting to select examples

We get to this step only if we could not find three problems that fall within the student's zone of proximal development. In this step, the examples that are more probable to be understood by the student are selected first, as they are placed on the top of the ranked list of examples. Note that the recommendation approach does not recommend an example if the student has solved all of the challenges related to that example. Because we assume that if the student has solved all of the challenges related to an example, the student already knows that example well and, as a result, work with that example will not help the student to learn more.

Step 4: Attempting to select the problems

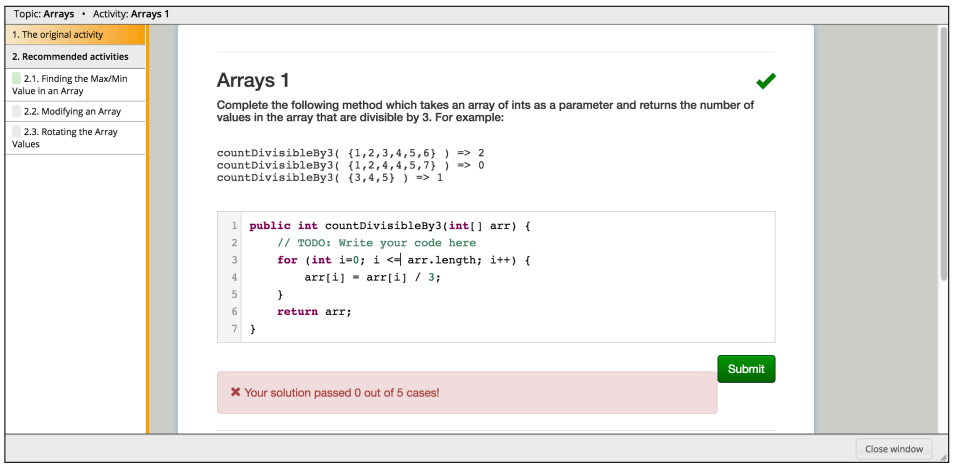
We get to this step, only if we could not complete the list of recommended activities in the previous steps. The idea here is to select the simplest activities first. Because when we get to this step, it is likely that the student was not ready to attempt problems and also the number of examples that could be recommended was not sufficient to complete the list of recommendations. Therefore, since challenges require less effort to

solve than the coding exercises, we attempt to recommend them first. If after searching the ranked list of challenges we still need more activities to complete the list of recommendations, then the we select the remaining activities from the ranked coding exercise list.

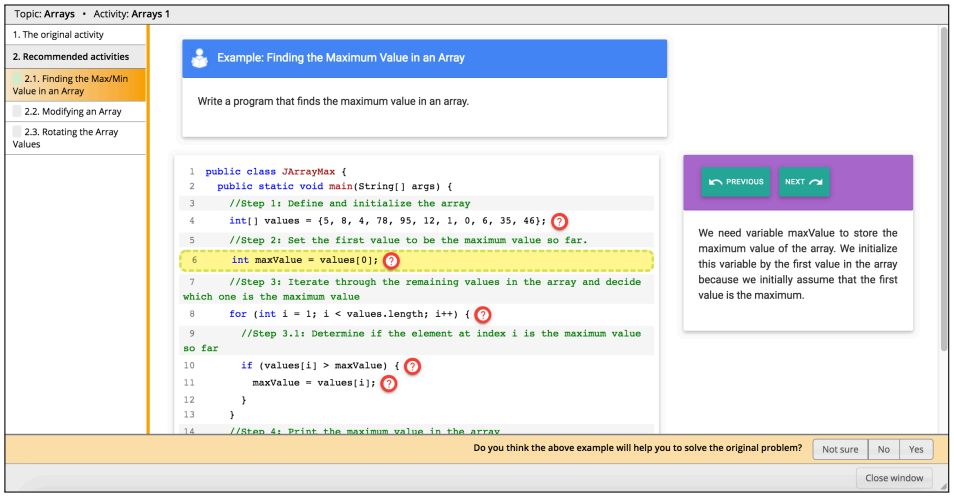
8.2.2 Reactive recommendation

Reactive recommendations occurred only when the student failed in solving a problem (i.e., a challenge or a coding exercise) correctly in the practice system. When the student failed to solve a challenge or a coding exercise, the reactive recommendation approach suggested top–3 most relevant remedial examples to the student. The example recommendation interface is shown in Figure 28(a), where the left panel shows the top–3 recommended examples to the student after she/he failed in solving the given coding exercise. The student can select any of the recommended examples and explore the explanations (Figure 28(b)). The recommendation interface is the same for challenges that student could not solve.

In our prior work in [Hosseini and Brusilovsky, 2017], we investigated different approaches for recommending examples that might be helpful for students who failed to solve a programming problem. As the key factor for generating recommendations, we considered similarities between the target problem and different examples. To determine the best recommendation approach, we explored a range of concept-level similarity approaches that assessed the similarity of problems and examples in terms of programming concepts measured within small fragments (structures), as well as within the content as a whole. In addition, we explored the value of considering student factors, such as student's knowledge and learning goals, as defined by the student's position in a course. Among the approaches explored in [Hosseini and Brusilovsky, 2017], an approach that took into account both the student's knowledge and goal delivered excellent performance in the expert evaluation and was comparable with other top approaches in the user evaluation. This approach selected the most helpful examples for a problem by



(a)



(b)

Figure 28: Part (a) shows the recommended examples for the current problem that student could not solve. 'The original activity' and 'Recommended activities' on the left panel provide access to the problem and the list of recommended examples, respectively. Part (b) illustrates a recommended example selected by the student. Student could navigate through the explanations in the examples.

(a) focusing on the examples that were similar to the problem by the structure of the concepts rather than concept coverage (i.e., being a structural similarity approach), (b)

personalizing example selection to select examples that matched student's knowledge levels and learning goal (i.e., being a personalized similarity approach), and (c) calculating the concept-level similarity of the examples to a problem using a cosine metric. We used this approach to generate the reactive recommendations in this study.

8.3 THE STUDY

8.3.1 Hypotheses

The study tested the following hypotheses regarding the benefits of *adaptive* recommendation of learning activities relative to *random* recommendation of learning activities:

*H*1. *Adaptive* recommendation approach would lead to greater time-on-task and interactions with the learning activities than the *random* recommendation approach

*H*2. *Adaptive* recommendation approach would lead to better learning outcomes than the *random* recommendation approach

8.3.2 Study design

The study had a 1×2 factorial design with the between-subject factor "recommendation approach" (*Random*, *Adaptive*). Students were randomly assigned to the Control and Experimental groups at the beginning of the semester. In the Control group students received recommendation using the *random* approach while in the Experimental group students received recommendations using the *adaptive* approach. In the Control group, both proactive and reactive recommendations were generated randomly: proactive recommendations were generated by randomly selecting the learning activities within each topic and reactive recommendations were generated by randomly selecting top–3 examples from the current topic or previous topics. In the Experimental group students

received proactive and reactive recommendations using the approaches described in Sections 8.2.1 and 8.2.2, respectively.

8.3.3 Participants and procedure

The study was conducted during the Spring 2018 semester. Participants were 205 undergraduate students with Computer Science majors who were enrolled in the CS0401 Intermediate Java Programming course at the University of Pittsburgh. This course was meant to be a second course in programming. The prerequisite was prior programming experience, ideally in Java. The course spent the first month reviewing nearly all the topics of introductory Java programming course. The remainder of the course focused heavily on Object-Oriented Programming with several significant projects using inheritance, implementing interfaces, abstract classes, inner classes, GUIs and event-driven programming.

At the beginning of the semester, all the students were provided with individual accounts to access the practice system (similar to the one described in Section 5.3). Students were first asked to take the online pre-test during a lab that was held during the first week. After students completed the pre-test, the system was introduced and students were told that the usage of the practice system is voluntary. The post-test was conducted one week before the final exam. After taking the post-test, the students were asked to answer to two online surveys that asked about examples and challenges (using survey described 4.2.4.1) and recommendations (using the survey described in 4.2.4.2) in the practice system.

To encourage participation in the pre-test, post-test, and survey, students received lab points for completing them. The usage of the practice system was voluntary. To encourage students to use the practice system, they were told that they would receive 3% extra credit upon completion of the minimum number of activities inside the practice system. The minimum amount of practice was completing 3 *PCEX* examples, 5 *PCEX* challenges, and 7 PCRS coding exercises.

8.3.4 Materials

8.3.4.1 Practice content The practice contents included 55 *PCEX* worked examples (as in Figure 6), 76 *PCEX* challenges (as in Figure 7), and 46 coding exercises served by PCRS tool (as in Figure 10). The practice system included 14 topics covering the topics for the first three weeks of the course which was a review of topics in the Introductory Programming. As shown in Figure 26, in each topic, there were three different types of smart content available: 1) worked examples that had the presentation style that was the same as worked examples in the *PCEX* tool (Figure 6), 2) challenges that had the presentation style that was the same as challenges in the *PCEX* tool (Figure 7), and (3) problems which were presented using the PCRS system (Figure 10). Note that unlike Classroom Study 1, the examples and challenges were shown in separate rows of the practice system interface. We made this change to make the recommendations for individual challenges related to a *PCEX* activity possible.

8.3.4.2 Pre- and post-tests The pre-test and post-test were online tests that were different but isomorphic and consisted of 10 questions. The first 5 questions were comprehension questions that asked the student about the value of a variable after the code was executed or the program output. After that, they had to answer 5 coding questions that asked the students to write the code using the PCRS tool. The estimated time to answer pre-test and post-test was 20 minutes but students could use the whole lab hour. They were asked to complete the tests without seeking any extra help and that they would not be penalized for any wrong answer.

8.3.5 Metrics

We grouped our metrics into engagement and learning metrics, as described in Section 4.2. The engagement metrics included measures focused on overall engagement and also persistence in working with activities. The overall engagement metrics included number of distinct attempts on each type of activity (examples, challenges, and coding exercises)

as well as measures focused on the amount of work with recommended activities: (1) distinct attempts on the recommended activities overall and on recommended examples, challenges and coding, and (2) follow rate, which was defined as the percentage of student's attempts that were on recommended activities.

We measured student's persistence in working with examples by using median example line clicks and median time on examples. For measuring persistence in solving problems, we used: median attempts on problems that the student could not solve; persistence probability, which was defined as ratio of problems that the student persisted on solving the problem after the first incorrect attempt to the problems that student failed in the first attempt; and the probability of not solving a problem.

We measured student's learning using pre-test, post-test, and midterm scores. We also looked into learning efficiency, which was defined as the number of problems the student solved (including challenges and coding exercises) to get 1 point increase in the normalized learning gain (as described in Section 4.2). As this was a very small number, we multiplied this number by 100 to express it as a percentage.

8.4 RESULTS

After the data collection was completed, we had 122 students who had at least one activity during the semester in the system (61 in Control, 61 in Experimental). Among them, 23 obtained extremely high pre-test score (i.e., 8.5 or above out of 10). We discarded the data of those 23 students since they had little to learn and were likely only participating for extra credit. The final dataset included the data from 99 students: 48 students in the Control group and 51 students in the Experimental group. We used this dataset for examining the overall usage and performing the engagement analysis. The groups were balanced in terms of prior knowledge, as the mean pre-test score in Control ($M = 4.5, SD = 2.4$) was not significantly different from the Experimental group ($M = 4.6, SD = 2.6$), using the One-way ANOVA test ($F(1, 96.9) = 0.02, p = .885$).

For the analysis related to midterm, we limited the final dataset to the data of 87 (41 in Control, 46 in Experimental) students who had taken the midterm and used the system before the midterm. For the analysis related to post-test, we excluded data of 16 students as their post-test score was lower than their pre-test score — indicating that they did not take the post-test seriously and most likely took it only to get the lab points. After discarding those students, we had data of 83 students (39 in Control, 44 in Experimental) for the post-test analysis[1].

The following subsections present the results from the data analyses. Table 20 summarizes our results by showing the hypotheses of Classroom Study 3 (described in Section 8.3.1), the corresponding data analyses, and whether the hypotheses were confirmed by the data analyses.

Table 20: Summary of hypotheses and results of Classroom Study 3.

Hypotheses	Data Analyses	Measures	Hypotheses Confirmed? Group effect	Hypotheses Confirmed? Interaction effect
H1 – adaptive vs. random recommendation: engagement	Section 8.4.1 and 8.4.2	overall engagement	✓(only for examples)	N/A
		median example line clicks	x	N/A
		median time on examples	✓	N/A
		median attempts on problems not solved	x	N/A
		persistence probability	x	N/A
		probability of not solving a problem	✓	N/A
H2 – adaptive vs. random recommendation: learning	Section 8.4.3	midterm	x	✓
		post-test	x	x
		learning efficiency	x	x

8.4.1 Overall engagement analysis

The first hypothesis ($H1$) was that *Adaptive* recommendation approach would lead to greater time-on-task and interactions with the learning activities than the *random* recommendation approach. To test this hypothesis, we compared the groups in terms of

[1]When all student data were included in the post-test analysis, pre-test score was the only significant predictor of the post-test score.

Table 21: Means (and SD) for usage of learning activities in Control and Experimental groups, along with inferential statistics for the group contrast.

	Control (N=48)			Experimental (N=51)			
	Median	Mean	SD	Median	Mean	SD	Wilcoxon test
OVERALL							
Dist. activities	24	31.1	27.4	22	33.2	31.6	$W = 1226.5, p = .989$
Dist. example attempts	7	11.5	11.6	8	12.9	11.9	$W = 1137, p = .544$
Dist. challenge attempts	8.5	12.4	12.4	8	15.1	17	$W = 1207.5, p = .911$
Dist. challenges solved	7	11.1	12.4	7	14.1	17.2	$W = 1207, p = .908$
Dist. complex challenge attempts	4	5.7	5.7	4	6.4	8.1	$W = 1288, p = .655$
Dist. complex challenges solved	4	5	5.6	4	6	8	$W = 1245, p = .885$
Dist. coding exercise attempts	7	7.1	6.4	2	5.2	7.2	$W = 1539.5, p = .026^{*}$
Dist. coding exercise solved	6	5.8	5.8	1	4	6.5	$W = 1573.5, p = .013^{*}$
Dist. coding exercise partially solved	6.5	6.2	5.9	1	4.5	6.8	$W = 1560, p = .017^{*}$
Dist. complex coding exercise attempts	0	1.5	2.4	0	1	2.1	$W = 1374, p = .245$
Dist. complex coding exercise solved	0	1.1	2.3	0	.7	2	$W = 1385, p = .171$
Dist. complex coding exercise partially solved	0	1.2	2.3	0	.8	2	$W = 1351, p = .306$
RECOMMENDED							
Dist. activities	13.5	15.8	11.5	11	20.4	21.8	$W = 1183, p = .777$
Dist. example attempts	4	4.3	3.3	5	6.8	6.4	$W = 946, p = .051$.
Dist. challenge attempts	6	6.8	5.3	5	10.6	12.1	$W = 1156.5, p = .638$
Dist. challenges solved	5	6	5	5	9.9	11.7	$W = 1152, p = .616$
Dist. complex challenge attempts	3	3.5	3.1	3	5	6.1	$W = 1180, p = .759$
Dist. complex challenges solved	2	3.1	3	3	4.6	5.7	$W = 1131, p = .513$
Dist. coding exercise attempts	4	4.7	5.3	0	3.1	5.8	$W = 1685.5, p < .001^{***}$
Dist. coding exercise solved	3	3.6	4.8	0	2.5	5.5	$W = 1645, p = .002^{**}$
Dist. coding exercise partially solved	3	3.9	4.9	0	2.8	5.8	$W = 1664, p = .001^{**}$
Dist. complex coding exercise attempts	0	1	2.2	0	.7	2	$W = 1388, p = .167$
Dist. complex coding exercise solved	0	.9	2.1	0	.5	1.9	$W = 1342, p = .273$
Dist. complex coding exercise partially solved	0	.9	2.1	0	.6	2	$W = 1355, p = .243$

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

overall engagement metrics, described in Section 8.3.5, which included the total work on activities, the rate students followed recommendations, and the work on recommended activities.

There were no significant differences in the mean follow rates for the two groups using one-way ANOVA test ($F(1, 97) = 1.46, p = .231$). The follow rate had a normal distribution with a mean of .5 in both groups, suggesting that both groups followed the recommendations, on average, in half of their attempts. Table 21 shows the information about the usage of the learning activities overall, and usage of recommended learning activities in both groups. Overall, the two groups did not differ significantly in terms of the median of total distinct example or challenge attempts. Students in the Control group attempted and solved significantly more distinct coding exercises. Yet, the number of distinct attempts on complex coding exercises and the number of distinct complex coding exercises solved was not significantly different in the two groups. When looking only at the distinct recommended attempts, the students in the Experimental group had marginally higher distinct attempts on the recommended examples. The number of distinct recommended coding exercises attempts and distinct recommended coding exercise solved was significantly higher in the Control group.

In sum, the comparison of the two groups in terms of overall engagement supported Hypothesis $H1$ by showing that the students in the Experimental group were more engaged in working with the recommended examples. While students in the Control group attempted and solved more problems, there were no differences in terms of the number of complex coding exercises that the student solved.

The differences between the usage of learning activities in the two groups suggest that the recommendation approach in the Control group guided students to simpler coding exercises while the recommendation approach in the Experimental group did not. Instead, the recommendation in the Experimental group encouraged students to work more on the examples, a learning activity that is usually neglected by students (at least by advanced students in the class such as ours). This hints that recommendations made by adaptive approach were able to change the behavior of students in a positive manner by guiding them to work with examples and not recommending them simple coding exercises that were unnecessary for them. These differences in the recommended

Table 22: Descriptive statistics for metrics representing engagement on recommended examples in Control and Experimental groups, along with inferential statistics for the group contrast.

	Control (N=47)			Experimental (N=49)			
	Median	Mean	SD	Median	Mean	SD	Wilcoxon test
Median example line clicks	7	6.3	4.5	5.8	5.3	3.8	W = 1039.5, p= .680
Median time on example (sec)	24	54.9	74.2	21	48.5	56.1	W = 1049.5, p = .622

attempts also suggest, but do not provide fully definitive evidence, that the adaptive approach in the Experimental group was able to adapt to the student's knowledge.

8.4.2 Persistence analysis

We further tested ($H1$) by examining the differences between the two groups in terms of persistence measures, described in Section 8.3.5. We used the Wilcoxon test to compare the group means for the persistence metrics representing how much the student persisted in working with a recommended example and in solving a recommended problem/challenge.

Table 22 shows the group means for the measures representing how much students were engaged in their work with the recommended examples. We found no significant differences on the mean time spent on examples ($p = .622$) or the number of lines clicks ($p = .680$) for the recommended examples in the Control and Experimental groups. We also compared the recommended examples and not recommended examples inside the Experimental group. We found no significant differences in the line clicks ($p = .322$) but students spent significantly more time on the recommended examples, about twice more than the time they spent on the examples that were not recommended ($p = .013$) (Table 23).

Table 23: Descriptive statistics for metrics representing engagement on not-recommended examples and recommended examples in the Experimental group, along with inferential statistics for the group contrast.

	Recommended examples			Not-recommended examples			
	Median	Mean	SD	Median	Mean	SD	Wilcoxon test
Median example line clicks	5.8	5.3	3.8	5	4.9	4.1	W = 844, p = .332
Median time on example (sec)	21	48.5	56.1	7.1	24	37.1	$W = 1255.5, p = .013^*$

$^*p < .05$

Table 24 shows the summary of engagement metrics on the recommended problems (challenges and coding exercises) in the two groups. The median attempts on problems not solved was not significantly different ($p = .204$). Similarly, there was no significant difference between the two groups in terms of persistence in solving the recommended problems ($p = .505$). The mean persistence probability was .9 in both groups, which indicates that both groups solved 90% of the recommended problems that they attempted. However, the probability of not solving recommended problems that the student attempted was about twice less in the Experimental than the Control group, and the difference was significant ($p = .016$).

Within the Experimental groups, we found a tendency for students having more attempt to solve problems. In particular, the median of attempts on problems not solved was about three times more in the recommended coding exercises than the not recommended coding exercises. However, the difference was not significant neither on this metric ($p = .146$) nor on the persistence probability ($p = .411$) or probability of not solving a problem ($p = .148$) (Table 25).

In sum, the persistence analyses supported Hypothesis $H1$, by showing differences in some of the persistence measures related to the work on examples and problems. In terms of persistence in problem-solving attempts, we found that the probability of not solving recommended problems in the Experimental group was twice lower than the rec-

Table 24: Descriptive statistics for metrics representing engagement on recommended problems in the Control and Experimental groups, along with inferential statistics for the group contrast.

	Control (N=44)			Experimental (N=46)			
	Median	Mean	SD	Median	Mean	SD	Wilcoxon test
Median attempts on problems not solved	3	4.3	3.5	3	3.4	2.8	$W = 546, p = .204$
Persistence probability	1	.9	.1	1	.9	.2	$W = 781, p = .505$
Probability of not solving a problem	.2	.2	.2	.1	.1	.1	$W = 1174, p = .016^*$

$^*p < .05$

Table 25: Descriptive statistics for metrics representing engagement on not recommended and recommended problems in the Experimental groups, along with inferential statistics for the group contrast.

	Not-recommended			Recommended			
	Median	Mean	SD	Median	Mean	SD	Wilcoxon test
Median attempts on problems not solved	0	0.1	0.1	3	3.4	2.8	$W = 416.5, p = .146$
Persistence probability	1	1	0.1	1	0.9	0.2	$W = 758.5, p = .411$
Probability of not solving a problem	0.1	0.2	0.3	0.1	0.1	0.1	$W = 970, p = .148$

ommended problems in the Control group. In addition, within the Experimental group, there was a tendency for having more attempts to solve recommended problems and also the probability of failure was lower on recommended problems than not recommended problems. In terms of persistence in work on examples, we did not find any difference between the work on the recommended examples in the Control and Experimental group. However, within the Experimental group, students spent significantly more time (about twice more) on the recommended examples than the not recommended examples. The results from the persistence analysis hints that the recommendations from the adaptive approach had higher quality compared to the non-adaptive approach.

8.4.3 Learning analysis

Our second hypothesis ($H2$) was the *Adaptive* recommendation approach would improve learning outcomes more than *Random* recommendation approach. To evaluate Hypothesis $H2$, We ran series of multiple regression analyses, listed in Table 26 to explore the following aspects of the impact of the system on first the student midterm and then post-test score in an iteratively deepening way:

- overall group benefit
- benefit of doing activities
- rate of following recommendations
- benefit of following recommendations
- benefit of doing more recommended activities

8.4.3.1 Impact of the system on midterm score Model 1 revealed an overall negative effect on the midterm for the Group factor ($F(2, 84) = 16.18$, $p < .001$, $R^2 = .28$). Being in the Experimental group was associated with a midterm score that was -4.5 ($SE = 2.6, p = .09$) lower than the Control group, at the same level of pre-test score. Model 2 showed no benefit for doing activities on the midterm score ($F(4, 82) = 10.96$, $p < .001$, $R^2 = .35$). Each distinct activity attempted was associated with .09 ($SE = .09$) increase in the midterm score which was not significant ($p = .317$). Yet, being in the Experimental group was associated with a midterm score that was 4.8 ($SE = 2.5, p = .062$) lower than the Control group, at the same level of pre-test score and at the same level of distinct activity attempts.

Model 3 did not show any relationship between the rate of following recommendations in the two groups ($F(3, 83) = 1.3$, $p = .279$, $R^2 = .05$). Model 4 showed no benefit for following recommendations ($F(5, 81) = 8.42$, $p < .001$, $R^2 = .34$) as neither follow rate ($p = .768$) nor the interaction of follow rate with group ($p = .752$) were significant

Table 26: Details related to the multiple regressions models that were performed for the learning analysis.

Model	The effect that is being tested	Independent variables	Dependent Variable
1	Overall group benefit	Group Pre-test	Midterm Post-test
2	Benefit of doing activities	Group Activity: Total distinct activity attempts Pre-test Group × Activity: interaction of Group with Activity	Midterm Post-test
3	Rate of following recommendations	Group Pre-test Group × pre-test: interaction of Group with Pre-test	Follow rate
4	Benefit of following recommendations	Group Frate: follow rate Activity: Total distinct activity attempts Pre-test Group × Frate: interaction of Group with Frate	Midterm Post-test
5	Benefit of doing more recommended activities	Group RecActivity: total distinct recommended activity attempts NoRecActivity: total distinct not-recommended activity attempts Pre-test Group × RecActivity: interaction of Group with RecActivity	Midterm Post-test

All numeric independent variables were mean centered to reduce potential multicollinearity problems [Aiken et al., 1991].

predictors of the midterm. Yet, we found that doing more activities was associated with a higher midterm score, when the effect of group, follow rate, pre-test, and the interaction of group and follow rate was controlled. Each additional distinct activity attempt was associated with a small increase of 0.2 ($SE = .06$) in the midterm score, ($p = .007$). Also, being in the Experimental group was also associated with a midterm score that was 4.8 ($SE = 2.6, p = .068$) lower than the Control group.

Model 5 showed the benefit of doing more recommended activities on midterm; $F(5, 81) = 9.26$, $p < .001$, $R^2 = .37$). Although the overall effect of the group on the midterm was negative ($Beta = -5.1, SE = 2.5, p = .047$), the interaction of the group and the distinct recommended activity attempts was positively related to midterm and close to being statistically significant ($p = .054$). More distinct attempts on the recommended activities in just the Experimental group increased the midterm score by .4 ($SE = .2$, $p = .054$) (Figure 29). The interaction of group and work with recommended learning activity remained to be significant when the model used only distinct attempts on recommended examples as its predictor ($Beta = 1.8$, $SE = .8$, $p = .023$). A similar trend was observed for the interaction between distinct recommended challenge attempts and group ($Beta = .9$, $SE = .5$, $p = .079$) which was marginally significant, and the interaction between distinct recommended coding exercise attempts and group ($Beta = .8$, $SE = .5$, $p = .132$) which was near-marginal significant.

8.4.3.2 Impact of the system on post-test score Model 1 revealed no overall group effect on the post-test score for the Group factor ($F(2, 80) = 35.05$, $p < .001$, $R^2 = .47$). Model 2 showed no benefit for doing activities on the post-test score ($F(4, 78) = 17.17, p < .001, R^2 = .47$). Model 3 showed that follow rates was marginally higher when the pre-test increased ($F(3, 79) = 3.01$, $p = .035$, $R^2 = .1$). The follow rate had an increase of .03 ($SE = .02, p = .076$) for each unit of increase in the pre-test score. There was no significant effect for group ($p = .781$) or the interaction of group with pre-test score on the follow rates ($p = .885$) . Model 4 showed no benefit for following recommendations ($F(5, 77) = 13.93$, $p < .001$, $R^2 = .47$) as neither follow

rate ($p = .387$) nor the interaction of follow rate with group ($p = .817$) were significant predictors of the midterm. No effect was found for the group on follow rates either ($p = .510$)

Model 5 showed negative effects, close to being marginally significant, for the group ($Beta = -0.67, SE = .4, p = .101$) and for the interaction of group with distinct recommended activity attempts effect ($Beta = -0.05, SE = .03, p = .140$), ($F(5, 81) = 9.26$, $p < .001$, $R^2 = .37$). There was a tendency for lower post-test scores in the Experimental group as well as doing more recommended activities at the Experimental group. At the same time, doing more recommended activities had an overall positive effect on the post-test score. Each additional distinct recommended activity attempts, regardless of the group, increased the post-test score by .06 ($SE = .03, p = .047$). And, interestingly, each distinct activity attempt that was not recommended decreased the post-test score

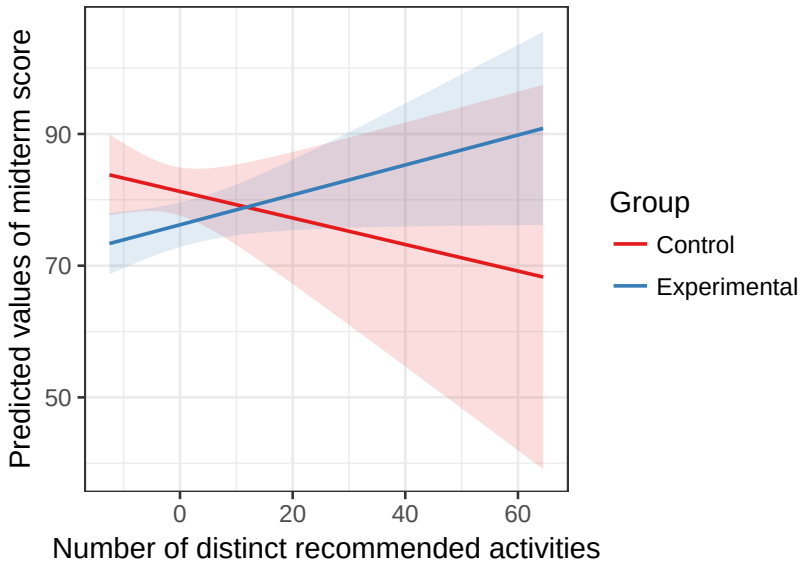


Figure 29: Interaction between distinct recommended activity attempts and Group factor (Control/Experimental) for predicting the midterm score. Notches indicate 95% confidence interval ranges.

by -0.03 ($SE = .02, p = .039$).

Overall, the post-test regressions results did not show any impact of the group or interaction of group and work with recommended activities on learning. But it should be noted that the usage data (Table 21) showed that students in the Experimental group did less work on the recommended coding exercises overall. This indicates that students in the Experimental group obtained the same amount of learning but with solving fewer problems. Therefore, we further examined the learning efficiency of the two group. The mean learning efficiency was higher in the Experimental group ($M = 8.7\%, SD = 18.6\%$) than the Control group ($M = 4.7\%, SD = 9.9\%$), yet the difference did not reach the significance ($W = 924.5, p = .480$). This is an interesting observation as it shows that despite the Control group solved significantly more coding exercises (overall, recommended), there was no significant difference in the learning efficiency. This, in turn, suggests that recommendations were smarter in Experimental group pointing the student to coding exercises with appropriate complexity level based on student's knowledge. As a result, students in the Experimental group obtained the same amount of learning with solving less coding exercises.

In sum, the multiple regression analyses partially supported hypothesis $H2$, demonstrating a positive impact of work with recommended activities in the Experimental group on the midterm score but not on the post-test scores. In the models predicting the midterm score, we did not find an overall effect for doing activity in (Model 2). However, in the previous studies we constantly saw the positive effect of doing activities. This is likely due to the study being conducted in an intermediate class where students were more advanced in programming compared to our previous studies. As a result, more distinct attempts does not bring any added value. At the same time, results of Model 5 showed that more distinct attempts on the recommended activities in the Experimental group increased the midterm score. This suggests that although the overall work with learning activities does not have an effect on the student's midterm in our

study, working on meaningful recommendations generated by the *Adaptive* approach does.

The regression model also showed a positive impact of the work with recommended examples on the midterm score inside the Experimental group. This is another confirmation of the value of examples. It also indicates that the *Adaptive* recommendations changed student's practice behavior in a positive way by encouraging them to view examples that were recommended, which in turn, resulted in higher midterm scores.

The regression models for the post-test showed a tendency (close to being marginally significant) for lower post-test scores in the Experimental group as students completed more recommended attempts. Overall, more work on the recommended activities, regardless of the group was associated with higher post-test scores, while more work with not-recommended activities was associated with lower post-test scores. The results of the post-test are inconsistent with the midterm results. We attribute this to the fact that post-test was performed too late, during the week before the final exam, when all students seemed to be in a good level of coding regardless of the amount of practice with the system. We think that this wrong timing could have caused the effects that we in observed in the midterm to be disappeared or reversed for the post-test. Given this, we think that midterm is a better measure of learning in our study compared to the post-test because it was taken in the 8th week of the class, where the student did not go very far from the basic topics that we covered in the system.

8.4.4 Analysis of reactive recommendations

Overall, reactive recommendations were not used extensively in this study. The number of total attempts in which the students could not solve the problems (challenges and coding exercises) was 944 in the Control group and 791 in the Experimental group. Among those attempts, the number of times that the students checked recommended examples was 58 (6%) in the Control group and 55 (7%) in the Experimental group;

which indicates a slightly higher usage of examples in the Experimental group. This is another important sign of better guidance in the Experimental group.

The low usage of remedial examples restricted us from performing any analysis to understand the impact of the *random* and *adaptive* reactive recommendation approach on student's learning. Yet, we ran some analysis to examine the group differences in terms of the number of remedial examples viewed; median line clicks on remedial examples; median time on remedial examples; revisits of the original problem after viewing the remedial examples; and solving the original problem after viewing the remedial examples. The results are shown in Table 27. Statistical analysis with unpaired t-test showed that, on average, number of recommended examples viewed was higher in the Experimental group than the Control group, and the difference was close to being significant ($p = .059$). No significant difference was observed between the Control and Experimental group in other measures.

In sum, higher usage of recommended examples in the Experimental group hints that students were more positive toward the recommendations generated by the *adaptive* approach which, in turn, resulted in more views of the examples. Although this is a sign for better performance of the *adaptive* approach, more data is needed to make concrete claims about the benefit of the *adaptive* approach over *random* approach for generating reactive recommendations.

8.4.5 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 one item from the quality construct because their item-construct correlations were lower than the recommended value, .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 it increased the internal consistency among the items of the construct, improving the α from .69 to .73. No item was discarded from the

Table 27: Descriptive statistics for usage of reactive recommendations

	Control (N=58)		Experimental (N=55)		
	Mean	SD	Mean	SD	t-test
Total examples viewed	2.1	1.3	2.7	1.9	$t(96.8) = -1.9, p = .059$.
Median line clicks	1.7	4	2.5	7.3	$t(83.4) = -.7, p = .48$
Median time on examples (sec)	14.2	30.7	13.4	30.9	$t(111) = .1, p = 0.89$
Revisit of the original problem	.5	.5	.5	.5	$t(111) = .7, p = .51$
Solving the original problem	.5	.5	.4	.5	$t(111) = .3, p = 0.76$

. $p < .1$

other constructs as all items had acceptable internal consistency with the other items within that construct. The α was .64 for the learning construct, .6 for the engagement construct, .84 for the quality construct, and .79 for the satisfaction construct. All of the alpha values appeared to be sufficiently reliable to assess the value of examples and recommendations, with α values exceeding the suggested minimum acceptable α coefficient of .50 [Nunnally and Bernstein, 1978].

In total, 151 students (71 in Control, 80 in Experimental) responded to the survey and provided consent to use their data. Among them, 20% ($N = 30$) used the system more than 10 times, 27% ($N = 41$) used the system between 5 and 10 times, 30% ($N = 45$) used the system less than 5 times, and 23% ($N = 35$) did not use the system at all. Overall, students mostly agreed that they did not use the system due to not feeling the need for additional help, preferring other resources and materials, and lack of time. Students disagreed with items that suggested other reasons for low/zero of the practice system, including the items that referred to bad system experience (Figure 30)

Figure 31 illustrates the mean ratings in the two study groups for the survey constructs that assessed the value of examples and recommendations (Table 28). As it can be seen from Figure 31, in both groups, students tended to agree on all construct except

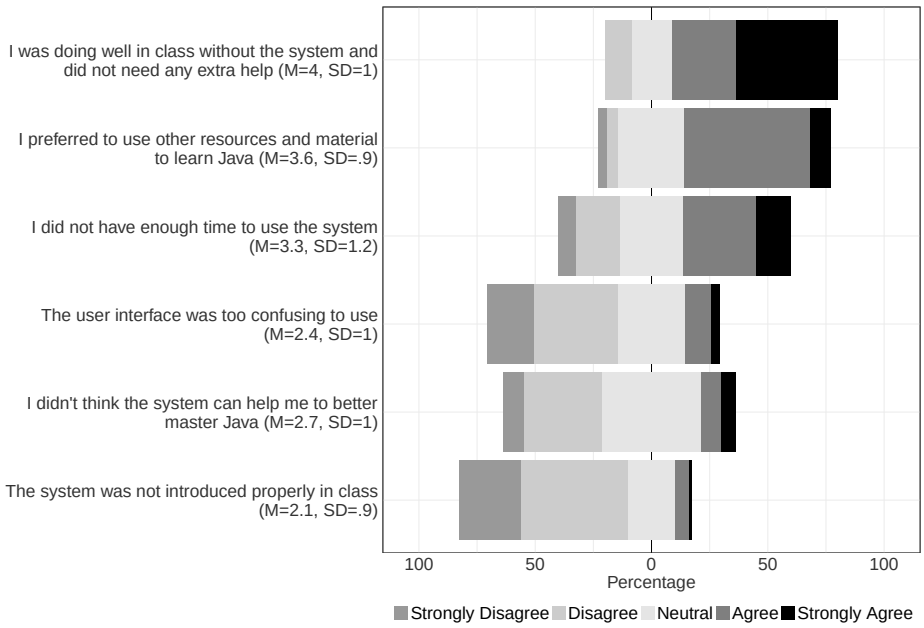


Figure 30: The distribution of answers for the survey items showing the reasons for not using the system. The percentage of respondents who agree/disagree with each item is shown to the right/left of the zero line. The percentage of respondents who neither agree nor disagree are split down the middle and are shown in a neutral color. The items in the y-axis are ordered based on the percentage of agreements, with the uppermost/lowermost item having the most/least agreement.

the engagement. The Kruskal-Wallis test showed no significant differences in the mean ratings of the study groups in the quality construct ($\chi^2(1) = 2.1, p = .147$), learning construct ($\chi^2(1) = .27, p = .605$), the quality of recommendations ($\chi^2(1) = .59, p = .444$), or system satisfaction ($\chi^2(1) = 2 = .65, p = .420$). The only difference in the mean construct rating between the two groups was in the engagement construct. The mean engagement rating was higher in the Control group ($M = 3.1, SD = .8$) than the Experimental group ($M = 2.8, SD = .9$), and the difference was close to significance ($\chi^2(1) = 3.82, p = .051$).

We further investigated the differences between the recommendation constructs for the students who were followers of recommendations. We considered a follower as the

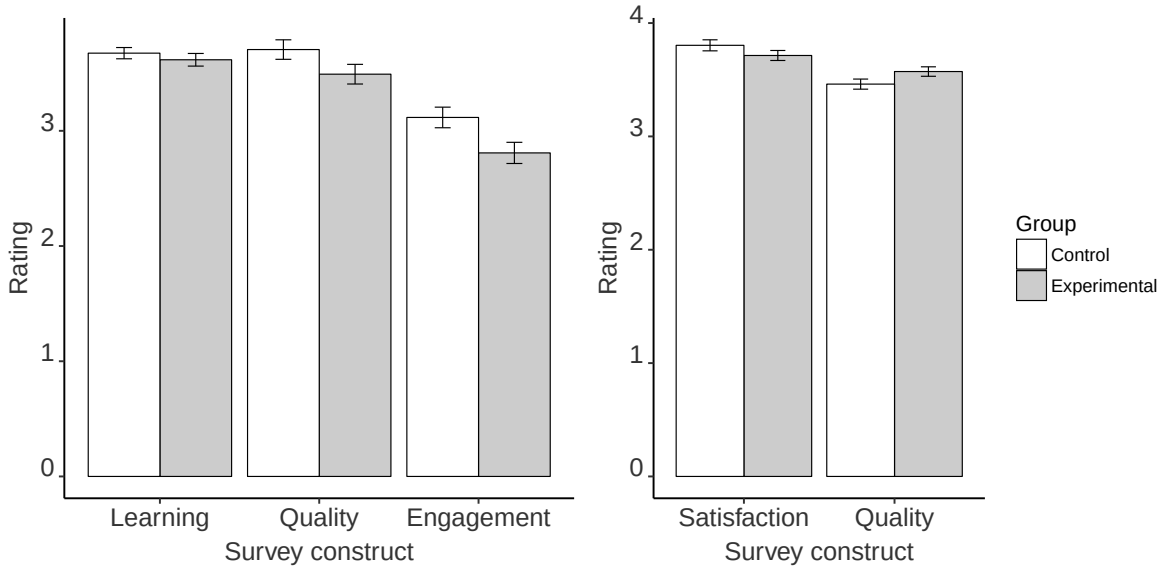


Figure 31: Mean and standard error of group ratings in the survey constructs. The left plot illustrates the mean ratings assessing the value of examples. The plot on the right illustrates the mean ratings assessing the value of recommendations.

student who had follow rate that was equal to or above the median of the follow rate. The median of follow rate was .5; therefore, a student was considered to be a follower, if she/he had followed recommendations in at least half of her/his attempts. By this definition, we had 57 students who were a follower, 32 were in the Control group, and 25 were in the Experimental group. Figure 32 shows the mean ratings for the recommendation and satisfaction constructs among the followers in the two groups. The followers in the Experimental group rated the quality of recommendation significantly higher than the followers in the Control group ($\chi^2(1) = 5.7, p = .017$). There was no significant difference in the mean rating of satisfaction construct ($\chi^2(1) = .05, p = .827$). We also examined the differences in ratings in the three constructs related to values of examples and found no significant difference for the learning ($\chi^2(1) = .672, p = .412$), engagement ($\chi^2(1) = .478, p = .490$), or quality construct ($\chi^2(1) = .0, p = .994$).

Table 28: The survey items assessing the value of examples and recommendations.

EXAMPLE EVALUATION
Quality
The explanations in the examples-challenges were not hard to understand [a]
The codes in examples-challenges were too complicated to understand[a]
Learning
The explanations in the examples helped me to better understand the Java programming concepts [a]
Working with the examples helped me learn Java
Exploring similar examples helped me learn Java
The examples helped me in solving Java exercises in this class [a]
Engagement
My mind was not wandering to other topics when I was looking at the examples- challenges[a]
I did not skim-read the examples [a]
RECOMMENDATION EVALUATION
Perceived Recommendation Quality
I liked the learning materials recommended by the system
The recommended learning materials fitted my needs
The recommended learning materials were well chosen
The recommended learning material were relevant to my goal
The system did not recommend too many bad learning materials [a]
I liked the recommended learning materials [a]
System Satisfaction
I would recommend the system to others
The system is not useless [a]
The learning materials that the system recommends are very helpful for me
The system recommendations help me have a better practice
Using the system is a pleasant experience
The system has real benefit for me [a]

[a] A reverse-coded item.

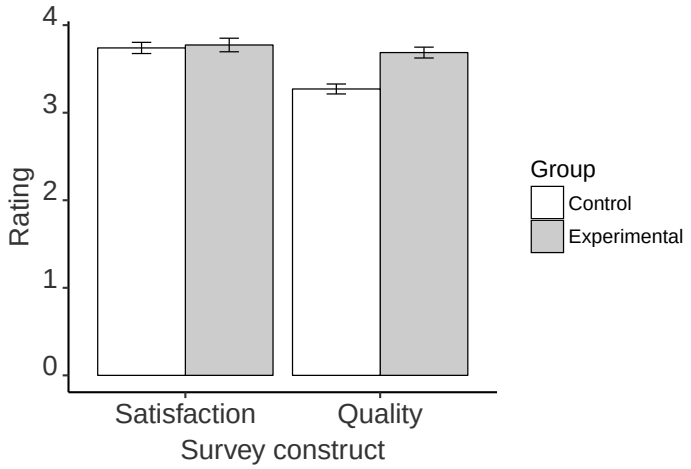


Figure 32: Mean and standard error of ratings of followers in each group in the survey constructs assessing the value of recommendations.

In sum, survey analysis showed no significant difference between the ratings of the two groups for the recommendation constructs. Followers in the Experimental group, on the other hand, rated the quality of recommendations significantly more than the followers in the Control group. There was no significant difference in the mean ratings of the followers for the satisfaction construct, though. The lack of difference in the satisfaction construct could be explained by the type of items in this construct, which targeted the system more than the recommendation.

8.5 SUMMARY AND DISCUSSION

8.5.1 SUMMARY

This chapter presented the controlled classroom study that was conducted to measure the effect of the adaptive recommendation of *PCEX* examples and problems on student's engagement in activities and learning. This section summarizes and discusses

our findings from the classroom study that compared adaptive recommendations to non-adaptive recommendations.

8.5.1.1 Effects of adaptive proactive recommendations Adaptive recommendations changed the the student's practice behavior in a positive way: We observed that the recommendations made by the adaptive approach guided the students to work with examples more than the random approach. Also unlike the random approach, the adaptive approach did not suggests students to stay on simple coding exercises that were not helpful for learning.

The adaptive recommendation made students more persistent to solve the problems that they attempted: Our analysis supported the engagement hypothesis ($H1$) by demonstrating positive effects of the adaptive recommendation on overall engagement and also on persistence in working with activities. We found that the probability of not solving problems recommended by the adaptive approach was significantly lower than the problems recommended by the random approach. Furthermore, when comparing the level of student's engagement on the recommended and not recommended activities within the Experimental group, we observed that the students spent twice more time on the recommended examples, which was significantly different from the time they spent on the not recommended examples. Additionally, the students tended to have more attempts to solve recommended problems and lower probability of failure in the recommended problems than the not recommended problems, and the difference was close to being marginally significant.

Adaptive recommendation partially influenced learning: Our analysis partially supported the learning hypothesis ($H2$). More distinct attempts on the activities recommended by the adaptive approach increased the midterm score, so did more distinct attempts on examples recommended by the adaptive approach. However, a rather opposite effect was found for the impact of adaptive recommendation on the post-test scores. We found that work with recommended activities was associated with getting a higher post-test score regardless of the approach that generated the recommendation.

At the same time, more work on the activities recommended by the adaptive approach tended to result in lower post-test scores (the effect was close to being marginally significant). And, more work on not recommended activities was significantly decreasing post-test score regardless of the group.

We explain this discrepancy between the midterm and post-test results by referring to the characteristics of the target population. The students in this study were enrolled in an intermediate programming class which spent more than 2/3 of the course on advanced programming concepts. It is likely that by the time of the post-test, the students had gained decent knowledge of programming and the concepts tested in post-test were too simple for them. As a result, the early effects that were seen at the time of mid-term disappeared at the end of the semester, when the post-test obtained taken.

8.5.1.2 Effects of adaptive reactive recommendations

Remedial examples were used in very few of the students' attempts that the students could not solve a problem (less than 10%). Therefore, not much could be claimed about the benefit of the reactive over random recommendations of examples. However, our results showed that the students were more positive toward the examples recommended by the adaptive approach as they viewed more examples (close to being statistically significant).

8.5.2 Discussion

Some of our findings need to be further discussed. First, we believe that the random approach in our study was a fair baseline and did not impair learning. The random proactive recommendations were selected from the learning activities within the topic that the student selected and the random reactive recommendations were selected from the examples that belonged to the current or previous topic of the problem that the student failed in solving it.

Second, the threshold we chose to determine whether the problems are within the zone of proximal development was selected by trial-and-error and pilot testing. This

limits us in generalizing our approach and findings to the other studies. Yet, we argue that our proposed adaptive strategy for selecting examples and problems based on student's knowledge could be easily applied to other domains.

Third, the findings from different learning measures were not consistent. Midterm was ideal in terms of the time it was taken during the semester but it was not isomorphic to the pre-test. Post-test was isomorphic to the pre-test but it was taken at the end of the semester when most of the students were advanced in programming; which faded away the impact of working with the recommended activities on student's learning. We could not use exam either as it was measuring concepts related to Data structures which were not covered in the practice system. Yet, we argue that the midterm is more accurate for measuring student's learning than the post-test because it was taken at the time that was close to the time that the review of Java basics was completed and students could use the system to practice those concepts. Moreover, in our models, we controlled for the effect of the prior knowledge using the pre-test score; which rules out any bias related to prior knowledge from our analysis. Therefore, we have enough evidence for partially accepting the learning hypothesis ($H2$).

Finally, the subjects in this study were not the ideal population because as they had prior experience in programming either in Java or another programming language. As a result, they did not feel the need for additional help and their behavior in the system was also a proof for that. In particular, when they failed in solving a problem, they often persisted in solving it without checking any of the recommended examples. Therefore, lack of usage data restricts us from drawing any conclusion about the benefit of the reactive recommendation approach over the random approach.

Overall, our study results provided enough evidence to conclude that the recommendations generated by the adaptive proactive approach were more effective for engaging the students and had some positive results on learning as well. However, we do see the need for conducting a similar study in an introductory programming class where students are in need of guidance which could be provided by the personalized approaches

in this study. Moreover, we suggest conducting this study in a larger class which could give us more statistical power for the regression analysis.

9.0 SUMMARY, CONTRIBUTIONS, AND FUTURE WORK

This chapter summarizes the findings of the studies described in previous chapters (Table 29), presents the contributions of this dissertation, points out the limitations of the studies conducted, and suggests directions for future research.

9.1 INTERACTIVE PROGRAM CONSTRUCTION EXAMPLES

This dissertation introduced *PCEX*, an interactive tool to support learning from program construction examples. *PCEX* made each program example explorable and engaging: students can explore each example interactively and check their understanding by solving challenges that are similar to that example. To promote learning, the examples in *PCEX* are enriched by worked steps with subgoal labels and explanations. To assess the impact of this new educational technology on student's engagement and learning (RQ1 – RQ4 in Section 1.2), this dissertation reports results from two semester-long classroom studies, Classroom Study 1, which explored how students would use *PCEX* examples and the relationship between the usage of *PCEX* examples and students' learning of programming (Chapter 5), and Classroom Study 2 which investigated the effect of *PCEX* examples on student's engagement and learning relative to non-interactive examples (Chapter 6).

Classroom Study 1, indicated that working on *PCEX* activities had a positive correlation with learning gain and student performance on problem-solving in coding

Table 29: Summary of hypotheses and results of the studies in this dissertation.

Hypotheses	Measures	Hypotheses Confirmed? Group/Condition effect	Hypotheses Confirmed? Interaction effect
CLASSROOM STUDY 2			
H1 – PCEX vs. textbook-style examples: engagement	time-on-task	✓	N/A
	interactions with explanations	x	N/A
H2 – PCEX vs. textbook-style examples: problem-solving	dist. Parson's problems solved	x	x
	assignment points	✓	x
	early submission	✓	✓
H3 – PCEX vs. textbook-style examples: learning	post-test	x	✓
	exam: code comprehension	x	x
	exam: basic code construction	x	x
	exam: complex code construction	✓	x
USER STUDY			
H1 – adaptive vs. no fading: practice problem-solving	number of attempts	x	✓
	problem-solving time	x	✓
	mental effort	x	x
H2 – adaptive vs. no fading: test problem-solving	problem-solving time	x	N/A
	efficiency	✓	N/A
	mental effort	✓	N/A
H3 – adaptive vs. no fading: learning	post-test	✓	N/A
CLASSROOM STUDY 3			
H1 – adaptive vs. random recommendation: engagement	overall engagement	✓(only for examples)	N/A
	median example line clicks	x	N/A
	median time on examples	✓	N/A
	median attempts on problems not solved	x	N/A
	persistence probability	x	N/A
	probability of not solving a problem	✓	N/A
H2 – adaptive vs. random recommendation: learning	midterm	x	✓
	post-test	x	x
	learning efficiency	x	x

exercises. Classroom Study 2 demonstrated that *PCEX* examples engage students more than static examples (similar to those presented in standard textbooks) do, by increasing their time on task and increase in activities. Specifically, we observed that students were less engaged in their work with regular examples than when working with *PCEX* examples, as indicated by the time spent practicing the *PCEX* worked examples and challenges. This is additional evidence in support of the underuse of the regular, static, textbook-like examples, mentioned earlier in Section 2.2.

Furthermore, *PCEX* examples resulted in several advantages over non-interactive examples, in terms of problem-solving performance: the group that practiced with *PCEX* examples obtained higher assignment points than the group that worked on static examples; and more work on *PCEX* examples was associated with early submission of assignments. Working with *PCEX* examples also resulted in an improvement in student's learning: practicing with *PCEX* examples had a positive impact on the student's post-test scores (near-transfer) as well as on the scores on exam questions that required a deep understanding of programming concepts, revealed when the student constructed a complex program (far-transfer).

The results from Classroom Study 2 showed several benefits for program construction examples. Working on examples (regardless of the technology used for presenting the examples) was associated with solving more Parson's problems correctly. Interestingly, this effect was even larger than the effect of the student's prior knowledge in predicting the number of Parson's problems solved correctly. Moreover, work on examples had a positive impact on exam scores, and the effect increased as the complexity of the exam questions increased. Our studies also provided insightful findings for explaining *when* and *how* the value of worked program examples are maximized. In particular, our results suggest that spaced/distributed practice with worked examples results in better learning outcomes than cramming practice right before an exam. The value of regular practice is also supported by modern research on spaced learning [Carpenter et al., 2012]. Our results also revealed a stronger effect for work with *PCEX* examples in the first half of the course than the second half, which reconfirms the findings of

previous research, which showed that the worked example effect is stronger in the early stages of learning and the benefit decreases as a student's knowledge grows [Kalyuga et al., 2003; Sweller et al., 1998].

9.2 PERSONALIZING WORK WITH PROGRAM EXAMPLES

To support individualized student work with *PCEX* examples, this dissertation study built and assessed technologies that adapted example presentation and selection to the student's knowledge. Example presentation was personalized, using an adaptive fading technology that adapted the amount of support provided within an example to the student's knowledge by adaptively fading steps in *PCEX* examples ("inner loop" adaptation). Example selection was personalized by using adaptive recommendation technologies to 1) guide the student to the most useful example or problems, based on her/his knowledge state and 2) to provide relevant examples to the student when she/he was having trouble solving a problem ("outer loop adaptation").

This dissertation reports on the results of two studies related to the area of personalization: A User Study (Chapter 7), which was conducted to examine the value of adaptive fading on student problem-solving performance and learning relative to non-fading example steps (RQ5 and RQ6 in Section 1.2), and Classroom Study 3 (Chapter 8), which investigated the value of adaptive recommendation technologies on student's engagement and learning relative to non-adaptive (random) recommendations (RQ7 and RQ8 in 1.2).

9.2.1 Adaptive fading of steps in program examples

The results from the User Study showed that adaptive *PCEX* examples that adapted fading of examples steps to the student's knowledge showed significant improvement on student problem-solving performance and learning over fully non-adaptive worked

examples with no faded step. Our results also showed the positive impact of adaptive fading on student problem-solving performance. As students progressed in their practice and completed more faded examples, they solved practice problems in significantly less time and marginally fewer attempts. Our results also showed that there was significantly lower mental effort invested in the problem-solving, a marginally higher problem-solving efficiency, and a tendency for a shorter problem-solving time that correlated with the adaptive fading of example steps. More importantly, adaptive fading was found to have an overall positive effect on the student's learning, as measured by pre-test and post-test.

9.2.2 Adaptive recommendation of program examples and problems

The results from Classroom Study 3 showed that adaptive recommendations changed the student's practice behavior in a positive way by guiding the students to work with examples when they needed it, instead of suggesting that students stay too long on simple coding exercises that didn't help them add to their programming knowledge. Secondly, our results showed that students with adaptive recommendation persisted more in working with the recommended examples and also in solving recommended problems. Specifically, the probability of the student not being able to solve problems that had been recommended by the adaptive approach was significantly lower than when the problems were recommended by a random approach. Also, students spent twice as much time on the adaptive recommended examples as they did on the examples that were not recommended.

Furthermore, adaptive recommendation partially influenced learning by demonstrating the positive impact of work on recommended activities on the midterm but not on the post-test. Our results showed that more distinct attempts on the adaptively recommended activities and examples increased the midterm score. However, the impact of adaptive recommendation on the post-test scores was not a strong factor. We found that working with the recommended activities was still associated with getting

a higher post-test score, but it didn't matter which approach generated the recommendation. And, more work on not recommended activities was significantly decreasing post-test score regardless of the group. This result seems somewhat surprising but is explainable by the characteristics of the studied population. Classroom Study 3 was conducted in an intermediate course and students were quite advanced at the time the post-test was taken. As a result, post-test was too simple for them and was not an appropriate test to measure the differences on student's learning at the end of the semester.

It should be mentioned that our results also showed that students were more positive toward remedial examples that were recommended to them adaptively. However, due to the limited scale of our usage data, we need to conduct a larger study in the future, to better understand the impact of the example-based problem-solving support (as used in Classroom Study 3) on student's learning and problem-solving performance.

9.3 CONTRIBUTIONS

This dissertation advances prior work in the area of building interactive program construction examples, designs a new learning technology tool, and studies their impact on student's engagement and learning. 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; Sweller and Cooper, 1985]). It was designed for the purpose of adaptively supporting individual students in the domain of programming. It helps them acquire program construction skills through exploring personalized and engaging worked examples. Our findings from these studies reconfirm that interactive, engaging worked examples help achieve better learning outcomes.

Our learning tool *(PCEX)* and the findings from Classroom Study 1 and Classroom Study 2 contribute to computer science education, the learning sciences, and worked-

example research by demonstrating the positive effect of *PCEX* examples on student engagement and the learning of programming, as well as showing some benefits for problem-solving performance.

The work in this dissertation contributes to the computer science community, in particular, the instructors of beginning CS courses, by creating more than 120 interactive program construction examples for introductory Java and Python. This is a valuable resource, as there is an emerging need for worked examples in the domain of programming. As mentioned in [Head et al., 2018], examples are often missing or insufficient for many programming tasks, and even when available, they may not be self-explanatory. On the other hand, *PCEX* examples, presented in this dissertation, are designed to be reusable so that instructors can easily add them to their course homepage or any online system that students use to practice on. These examples are also enriched with explanations and subgoal labels, to make understanding program construction easier for the students.

The work in this dissertation is an attempt to extend prior work on adaptive programming systems by introducing new technologies that are fine-tuned to the student's knowledge. Prior work, reviewed in Section 2.4, was not designed to address specifically how the student would transition from fully explained examples to faded examples and problems (e.g., [Guerra et al., 2018], [Weber and Brusilovsky, 2001], [Yudelson and Brusilovsky, 2005], [Davidovic et al., 2003], [Hosseini et al., 2015]). The adaptive recommendation and adaptive fading technologies in this work extend the existing line of research on adaptive educational systems in the field of programming, by addressing the aforementioned gap. On the one hand, an adaptive technology for proactive recommendations helps the student work on examples when she/he has none or very little knowledge of the subject, but as the student's knowledge grows, the system will guide her/him to move on to faded examples and later to problems. So, to solve this later demand, on the other hand, we have also created an adaptive fading technology. This dissertation describes how the step explanations in the worked program examples

can be gradually faded, based on what is appropriate for the student's current state of knowledge.

The User Study, described in Chapter 7, replicates prior research on faded examples (e.g., [Najar et al., 2016; Salden et al., 2009]) and confirms that the adaptive fading of example steps is helpful for problem-solving and learning. Our findings from this User Study align with previous research on adapting instruction to the student's knowledge by confirming research findings in the domains of Geometry [Salden et al., 2009] and SQL [Najar et al., 2014]. Moreover, they contribute to the fields of computer science education and AI in education, by providing insightful implications for how to scaffold program examples.

Classroom Study 3, described in Chapter 8, extends prior work on adaptive systems for guiding the student in her/his navigation to learning activities (e.g., [Guerra et al., 2018], [Weber and Brusilovsky, 2001], [Yudelson and Brusilovsky, 2005], Davidovic et al. [2003], Hosseini et al. [2015]) by demonstrating how and when examples, faded examples, and problems should be presented to the student in his/her practice sequence. Our goal was to replicate their findings within a new domain, programming. The findings from this study showed that the proposed adaptive technologies for recommending learning activities engaged students to be more persistent in each learning activity while simultaneously resulting in some positive results on learning achievement.

Finally, our findings from Classroom Study 3 contribute to computer science education and AI in education, by introducing a real-time online adaptive system for helping students learn to program. Recent online systems either support the student in only problem-solving (such as codingbat [Parlante, 2017], CloudCoder [Hovemeyer et al., 2013], CodeWorkout [Buffardi and Edwards, 2014]) or provided adaptive support only in problem-solving [Hsiao et al., 2010]. Only a few attempts offer practice with both examples and problems. Among them, we have non-adaptive E-books that offer example+practice problems [Ericson et al., 2015], as well as our prior work, which guided students to program behavior examples and problems, based either on progress [Hosseini et al., 2016] or an adaptive approach that aimed to maximize student's knowledge

[Guerra et al., 2018; Hosseini et al., 2015]. The system studied in Classroom Study 3 contributes to the existing online systems as well as our past work by using an advanced student modeling approach which offers personalized practice with program construction examples and problems.

9.4 LIMITATIONS AND FUTURE WORK

The work on this dissertation can be extended in several ways. The first direction for future work would be to improve the *PCEX* interface features. The second and third directions for future work would be to extend the studies (Section 9.4.2) conducted in this dissertation that assessed the impact of the examples and inner/outer loop adaptation for supporting student's work with *PCEX* 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.

9.4.1 Example design

In the current design, *PCEX* combines explorability and challenge in the same interface, which makes it impossible to assess the value of explorability and challenges separately. In the future, it would be good to conduct a controlled 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 hints and explanations were used very few times by the students. It is not clear why these features have not been used extensively. Thus, future research needs to run more usability studies for understanding how to improve the design of these features.

9.4.2 Studies assessing examples and adaptive technologies

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 Classroom Study 2. 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. 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 classroom studies, 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 before and after each practice session.

Although Classroom Study 2 was conducted in a rather large course, it was limited to a specific population. The target course in Classroom Study 2 was presented as a mandatory course for engineering students. Participants were mainly studying electrical engineering or civil engineering with a small a 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 is that there is a considerable portion of such non-CS students who are not well motivated to learn to program, especially when compared with computer science (CS) students. Thus, our

results from Classroom Study 2 may not generalize well for CS major students elsewhere, but may better generalize for CS minors or non-CS majors. Another limitation in Classroom Study 2 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 Classroom Study 2.

The User Study used a single control condition in which no example steps were faded. While this is a valid control condition and maximized the chance of registering the difference between the two conditions in our study, there are alternative methods such as fixed, backward or forward fading, which may be more suitable [Atkinson and Renkl, 2007]. Future work may extend the user study to investigate the impact of adaptive fading relative to these alternative approaches for fading example steps. Another limitation of the User Study is related to the duration of the study session, which may impact our results. Future work could conduct a study in multiple sessions, or alternatively in a classroom study. Faded examples in the user study were also presented without showing any explanations for the non-faded lines. This limits us in generalizing our findings to other domains, in which faded example only fade the explanations for steps not included in the solution. Therefore, a future study may investigate the value of adaptive fading when faded program examples hide the explanations only for the faded lines. Lastly, in the User Study we compared adaptively faded examples against standard examples to maximize the difference between the two conditions, thus, increasing the chance of registering differences between them. This is a compromise that doesn't allow a reliably measurable separation of the "fading" effects from the "adaptive" effects. Future work should improve the study design and increase the number of conditions in the study to have an additional condition with fixed fading which would allow us to separate the fading from the adaptive effects.

Classroom Study 3, which examined the effect of an adaptive recommendation for learning activities, was conducted in an intermediate programming course which re-

quired students to have prior experience in programming. As a result, in Classroom Study 3, the majority of the students were advanced in programming and did not need any additional support in terms of guidance. Therefore, the impact of the recommendation was probably not properly measured in this study. The characteristics of the population in this study also limited us in generalizing our findings to other populations, especially learners with little or no prior programming knowledge. Therefore, a similar study should be conducted with students in an introductory programming course, to more reliably assess the impact of adaptive recommendation on student's learning.

In Classroom Study 3, students were free to make their own navigational decisions, and as a result, in about half of their attempts, they did not follow the systems' recommendations. Free choice rather than sequencing was a design decision, and this limitation stems from it. We tried to minimize the effect of free choice by controlling for the effect of the number of student attempts on activities that were not recommended. However, we cannot guarantee that this effect could be fully controlled for as it is not straightforward how their work on not-recommended activities should be quantified. Should we take into account the concepts in those activities or the order they were accessed by the student or another factor? Future work may explore how students could be encouraged to follow recommendations while they have free choice. One promising approach would be to use gamification features in the interface, such as progress meters and winning certificates or badges.

Several studies suggest that adaptive systems should be more transparent to the users, that is the adaptive system should explain its decisions to the user. Classroom Study 3 lacks the transparency feature as it does not explain why it recommends the top–3 activities to the student. Thus, in the future, it would be interesting to examine the value of recommendations when the interface provides more information about its decisions to the students, explaining why it recommended each activity. One approach that could be explored would be to visualize how much each recommended activity could contribute to the student's knowledge.

One limitation in Classroom Study 3 is that only one adaptive recommendation approach was investigated. Although we found some positive effects to this approach, it's not yet clear whether this approach was more beneficial then other task selection approaches. In the future, we need to explore and assess more approaches. One approach for task selection would be to use state-of-the-art student modeling approaches such as performance factor analysis (PFA) [Pavlik Jr et al., 2009] to predict student performance on each activity and then determine the best activity should follow next, accordingly. Another approach would be to use recommendation approaches that take into account a range of student factors, rather than focusing on student's knowledge alone. Similarly, our results are limited to using a single approach in the control condition. Although having a random recommendation seems to be a fair baseline for within-topic recommendations of learning activities, there are other approaches which may be more suitable. Future studies should explore alternative baseline approaches such as a fixed sequence suggested by the teacher.

Another limitation of the adaptive recommendation approaches we used in the User Study and Classroom Study 3 dealt with the thresholds that we used in the adaptive fading and adaptive recommendation approaches. We adjusted the threshold for these approaches on a trial-and-error basis and through pilot-testing. Future research should determine these thresholds by running cross-validation on the data.

The adaptive technologies in the User Study and Classroom Study 3 could be investigated in a larger-scale study to evaluate the impact of having different adaptation levels used together in the same study. An ideal setting for such a study would be a 2×2 between-subject design with "outer loop" and "inner loop" as between-subject factors. This study would shed light on which combinations of adaptive educational technologies in program examples would lead to the best learning outcomes.

Finally, the Bayesian student model had some limitations. First, in programming, each activity is related to many concepts. On the other hand, the number of parameters that a Bayesian network can handle is limited. To tackle this issue, we followed the suggestion by Huang et al. [2014] and used the TF-IDF approach to determine which

of the concepts which were the most important for each activity. We kept about 30% of the concepts. Therefore, we lost precision in modeling, since we ignored 70% of the concepts in each activity. Another limitation is that we could not model the student's work on examples. Future work should explore possible ways of increasing the accuracy of the Bayesian student model in the domain of programming. Moreover, more research should be done to understand how student work with examples can be modeled.

9.4.3 Other directions

We plan to connect the work in this dissertation 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. Also, current adaptive recommendation technologies should be extended to determine how to provide cross-skill recommendations, i.e., when the student should be guided to a certain program behavior or program construction activities. Finally, we plan to build a repository of *PCEX* examples to make them available to the public. We also plan to develop an open source authoring tool to allow instructors and researcher to create and share *PCEX* examples.

9.5 DISCUSSION

A discussion of general issues in my dissertation is as follows:

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 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.

In this dissertation research, a time factor was used in several ways. In some places, more time spent on an activity was considered to be positive evidence for engagement while in other places, more time spent on an activity was considered to be negative, as evidence for having trouble completing the task. We argue that more time spent on an activity could be taken as a positive sign as long as it is coupled with better learning. Specifically, if more time spent on an activity does not improve learning achievement or even impairs learning, then it might merely be pointing to a bad interface design, bad content design, bad learning strategies in the study design, or bad habits employed by the learner. On the other hand, if more time spent on an activity results in improved learning, then it points to a good design of content or interface, indicating the student has become more engaged in working with activities.

Although, self-selection and free choice 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 our studies enabled us to investigate the impact of the proposed learning tool and the personalized technologies, in natural contexts.

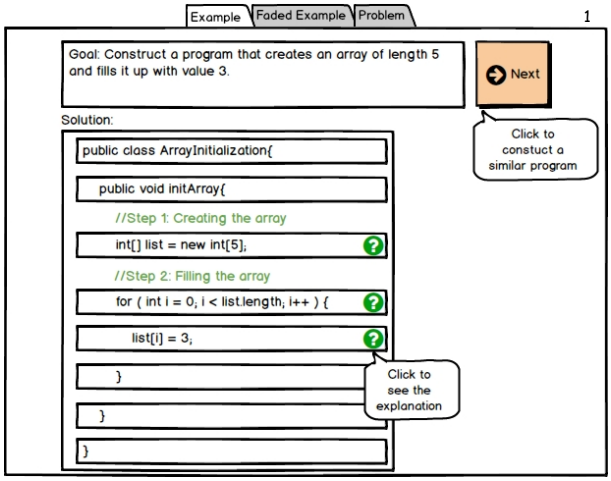
We acknowledged that Classroom Study 2 and 3 did not have the "best possible" student audience. Likewise, the Classroom Study 2 that we conducted in Finland helped us to investigate the impact of our system in the context of an educational system, but it was one that varied quite a bit from the educational system in the United States, as well

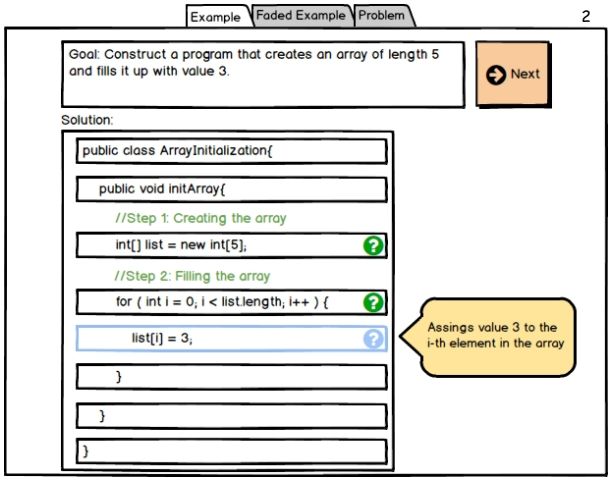
as from many other countries in the world. However, this should not be seen as merely a limitation because the studied population enabled us to see the impact our learning technologies had on more diverse groups of students (both beginners and intermediate learners; students in the U.S. and Finland). Similarly, the Classroom Study 3 that we conducted in intermediate programming classes enabled us to understand the impact of the proposed learning tool and adaptation technologies in the context of a class that requires students to have prior programming knowledge.

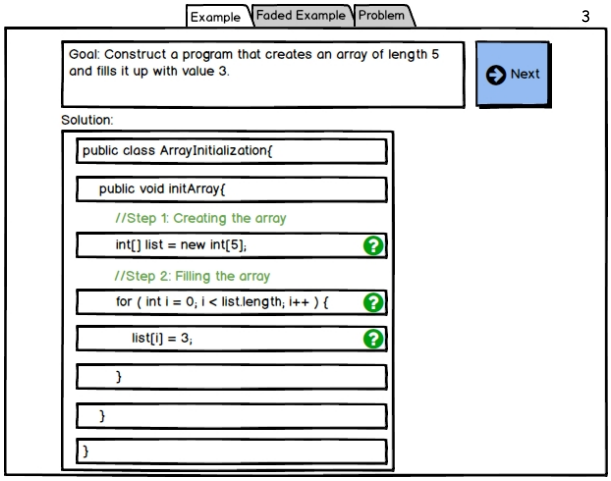
We observed a contradiction between a log-based and survey-based evaluation of engagement. The survey results showed that students reported low values for engagement with PCEX examples while the log-based data showed that students were engaged with the PCEX examples by spending more time on these examples compared to the regular examples. As mentioned earlier in Section 6.5.2, this contradiction between the survey and log-based data could be caused by these measures reflecting different constructs referred by the same name. 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, adaptive PCEX examples compared to studying a flat, static book, or the tutoring equivalent to a flat, static book. Future studies should conduct individual interviews with students to understand possible reasons for reporting lower engagement in the survey and also discuss options for enhancing engaging features in PCEX.

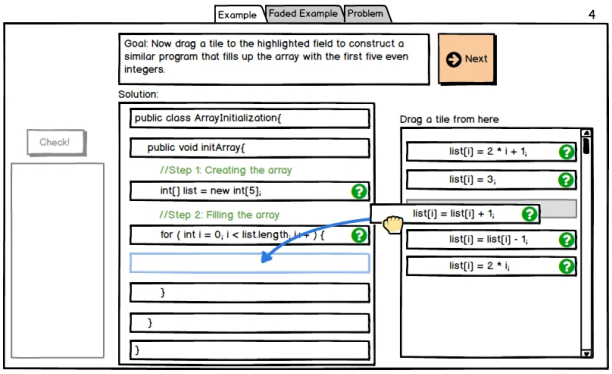
APPENDIX A

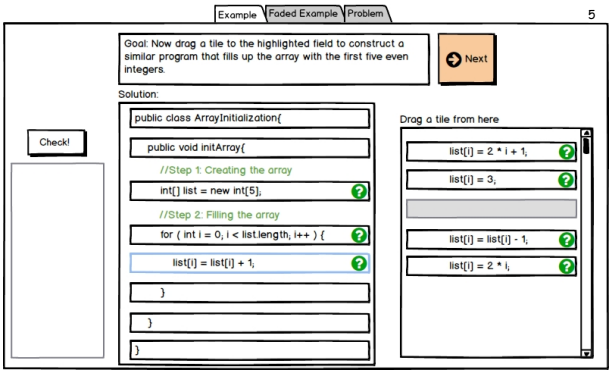
PCEX MOCK-UPS

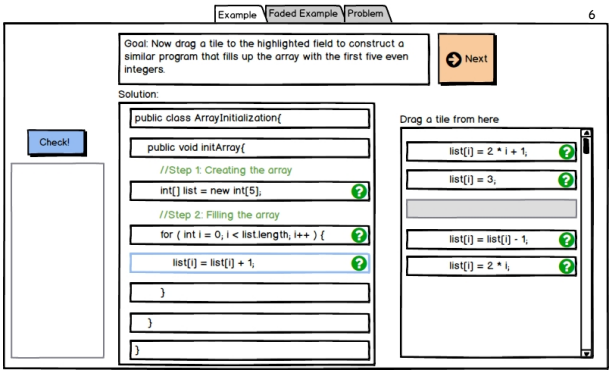

Example
Faded Example
Problem
1
Goal: Construct a program that creates an array of length 5 and fills it up with value 3.
Next
Click to constuct a similar program
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 3;
}
}
}
Click to see the explanation

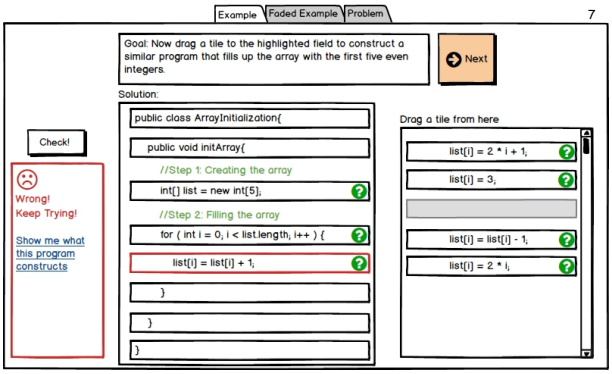

Example
Faded Example
Problem
2
Goal: Construct a program that creates an array of length 5 and fills it up with value 3.
Next
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 3;
}
}
}
Assings value 3 to the i-th element in the array

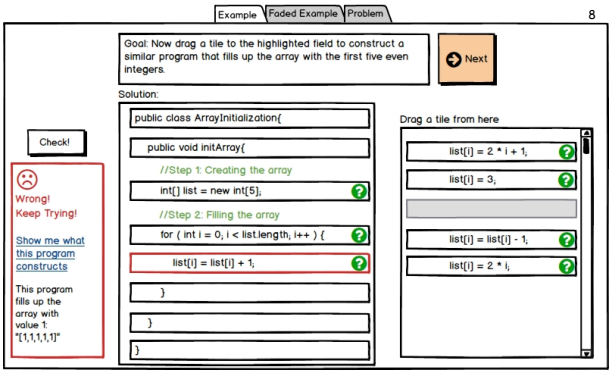

Example
Faded Example
Problem
3
Goal: Construct a program that creates an array of length 5 and fills it up with value 3.
Next
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 3;
}
}
}

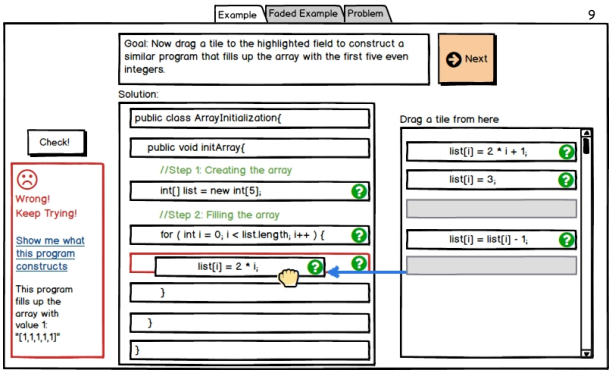

Example
Faded Example
Problem
4
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

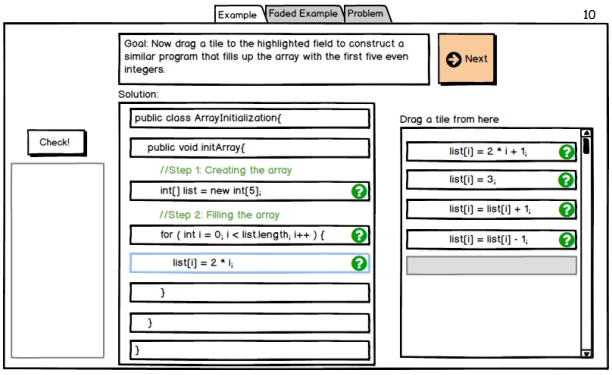

Example
Faded Example
Problem
5
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

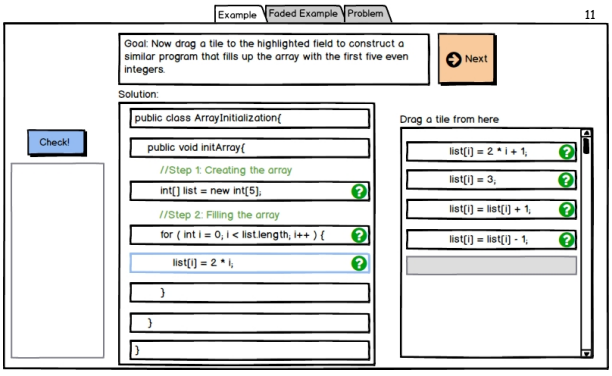

Example
Faded Example
Problem
6
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

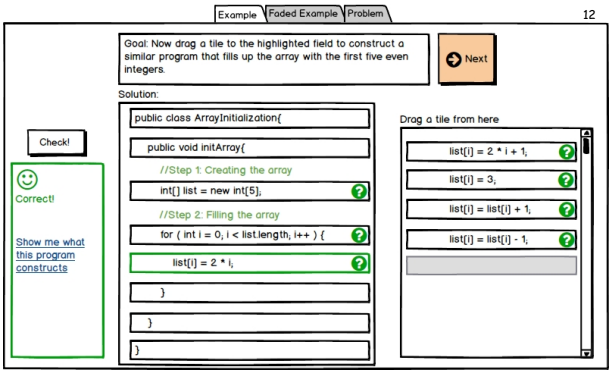

Example
Faded Example
Problem
7
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
Wrong!
Keep Trying!
Show me what this program constructs
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

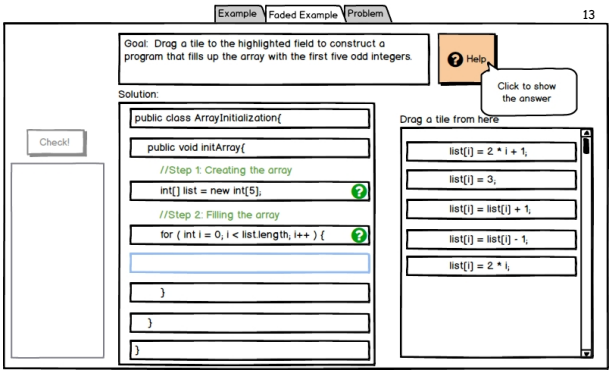

Example
Faded Example
Problem
8
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
Wrong!
Keep Trying!
Show me what this program constructs
This program fills up the array with value 1: "{1,1,1,1,1}"
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

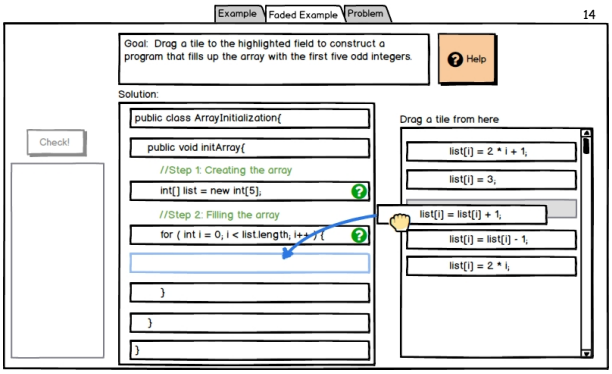

Example
Faded Example
Problem
9
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
Wrong!
Keep Trying!
Show me what this program constructs
This program fills up the array with value 1: "{1,1,1,1,1}"
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;

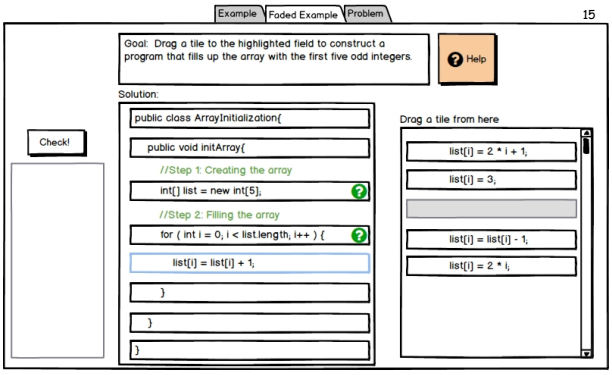

Example
Faded Example
Problem
10
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

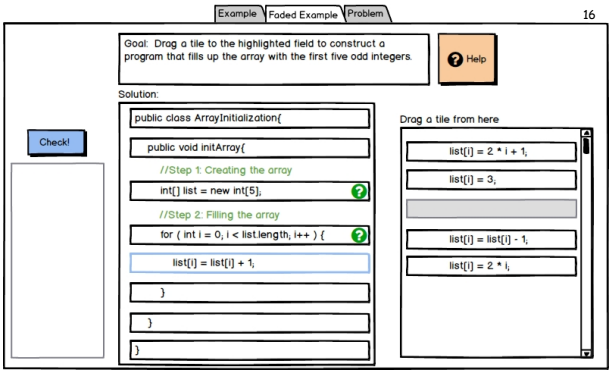

Example
Faded Example
Problem
11
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

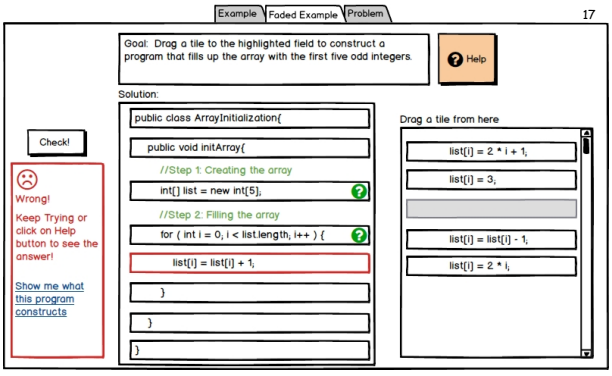

Example
Faded Example
Problem
12
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
Correct!
Show me what this program constructs
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

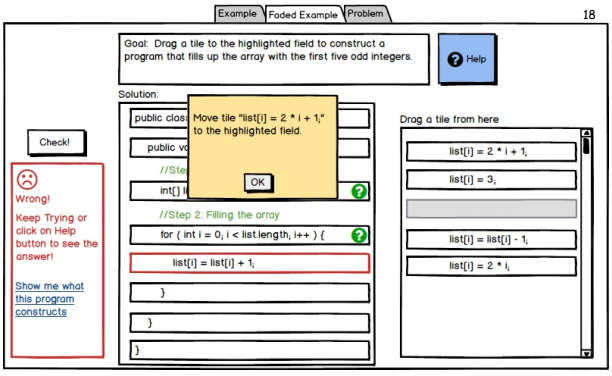

Example
Faded Example
Problem
13
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Click to show the answer
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

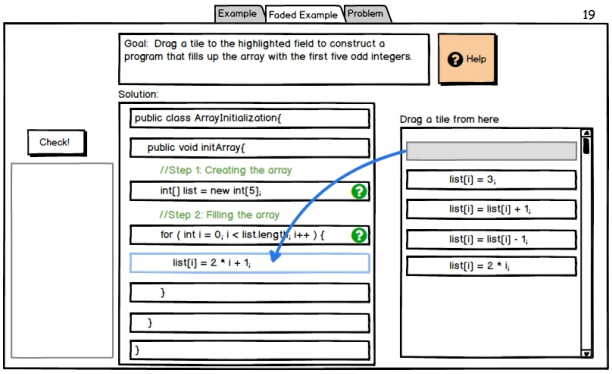

Example
Faded Example
Problem
14
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

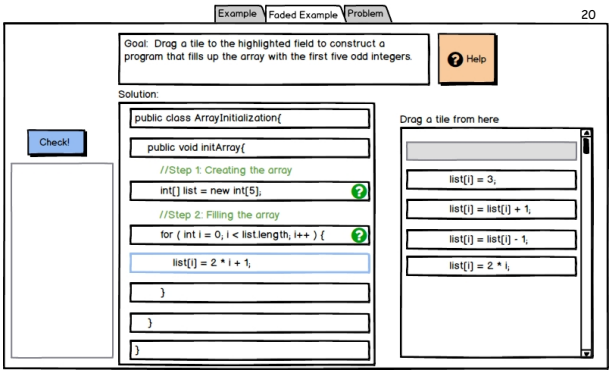

Example
Faded Example
Problem
15
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

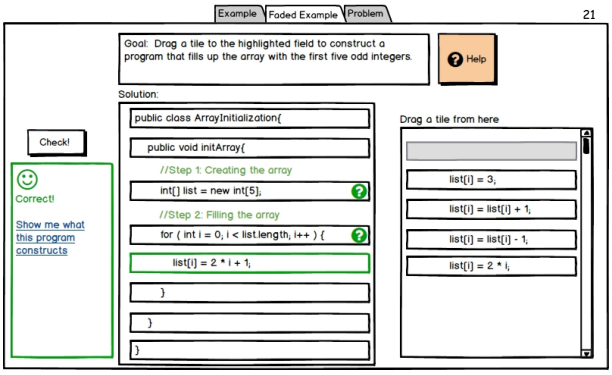

Example
Faded Example
Problem
16
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

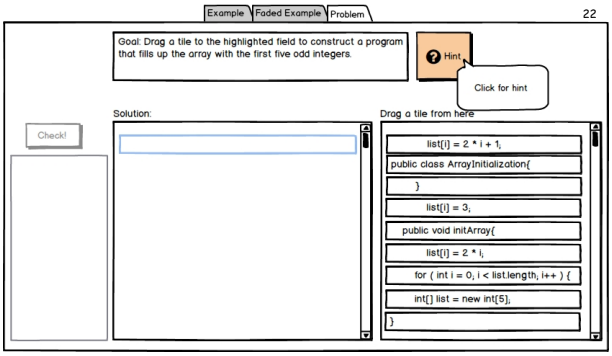

Example
Faded Example
Problem
17
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
Wrong!
Keep Trying or click on Help button to see the answer!
Show me what this program constructs
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

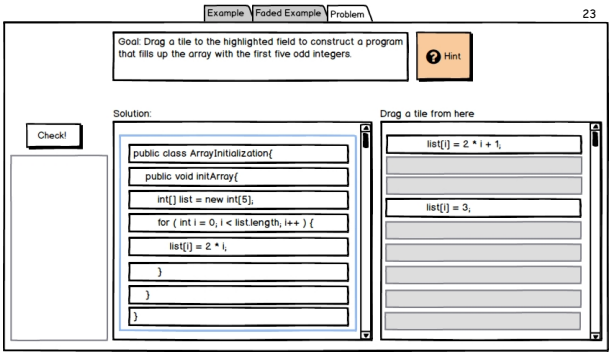

Example
Faded Example
Problem
18
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Move tile "list[i] = 2 * i + 1;" to the highlighted field.
OK
Solution:
Check!
Wrong!
Keep Trying or click on Help button to see the answer!
Show me what this program constructs
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

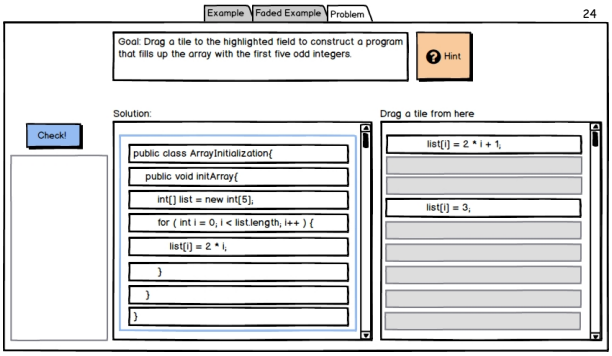

Example
Faded Example
Problem
19
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

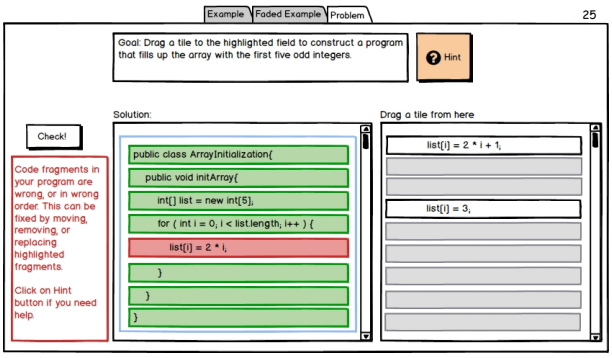

Example
Faded Example
Problem
20
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

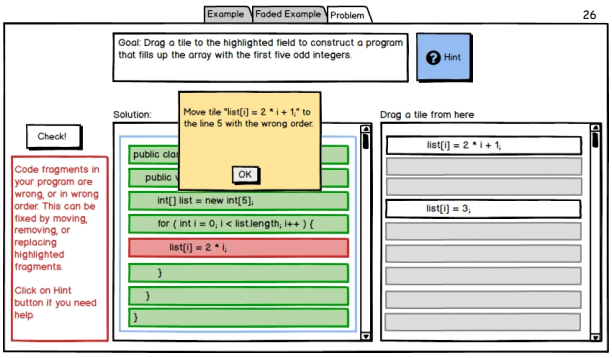

Example
Faded Example
Problem
21
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
Correct!
Show me what this program constructs
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

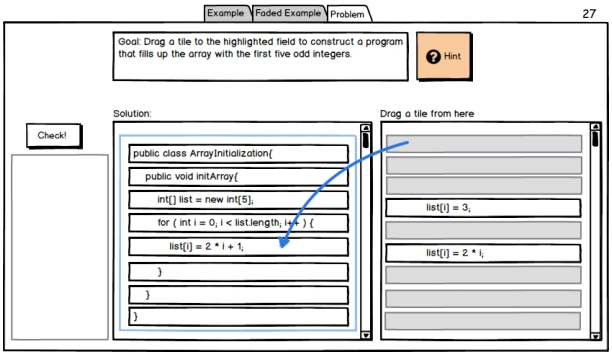

Example
Faded Example
Problem
22
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Click for hint
Check!
Solution:
Drag a tile from here
list[i] = 2 * i + 1;
public class ArrayInitialization{
}
list[i] = 3;
public void initArray{
list[i] = 2 * i;
for (int i = 0; i < list.length; i++) {
int[] list = new int[5];
}

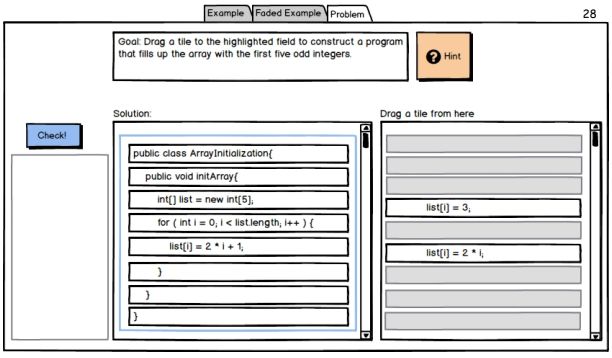

Example
Faded Example
Problem
23
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Check!
Solution:
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;

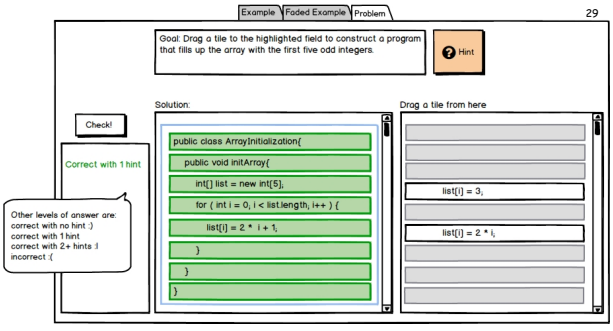

Example
Faded Example
Problem
24
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Check!
Solution:
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;


Example
Faded Example
Problem
25
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
Code fragments in your program are wrong, or in wrong order. This can be fixed by moving, removing, or replacing highlighted fragments.
Click on Hint button if you need help.
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;


Example
Faded Example
Problem
26
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
Move tile "list[i] = 2 * i + 1;" to the line 5 with the wrong order.
OK
Code fragments in your program are wrong, or in wrong order. This can be fixed by moving, removing, or replacing highlighted fragments.
Click on Hint button if you need help.
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
list[i] = 2 * i + 1;
list[i] = 3;


Example
Faded Example
Problem
27
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
list[i] = 3;
list[i] = 2 * i;


Example
Faded Example
Problem
28
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
list[i] = 3;
list[i] = 2 * i;


Example
Faded Example
Problem
29
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
Correct with 1 hint
Other levels of answer are:
correct with no hint :)
correct with 1 hint
correct with 2+ hints :|
incorrect :(
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
list[i] = 3;
list[i] = 2 * i;

APPENDIX B

PRE- AND POST-TESTS

B.1 PRE- AND POST-TEST IN CLASSROOM STUDY 1

The pre-test is presented below. The post-test was isomorphic to the pre-test: questions 1–3 were the same as the pre-test, questions 4–6 presented the given lines in a different order.

For questions 1 – 3 of this section complete the code by filling in the blank line(s) or box.

1. Complete the following code snippet to find the sum of the integer numbers entered from the keyboard. Assume that user enters 0 to indicate the end of input.

```
int sum= 0;
Scanner input=newScanner(System.in);
int num=input.nextInt();
while(..................){
    ..................
    num=input.nextInt();
}
System.out.print("Sum:"+num);
```

2. Complete the following code snippet to print the following output:

```
*
**
***
****
for(.................) {

 for(.................) {
   System.out.print("*");
 }
 System.out.println();
}
```

3. Write a method called firstTwo to receive a string str and return the string made of its first two chars, so the String "Hello" yields "He". If the string is shorter than length 2, the method returns whatever there is, and the empty string "" yields the empty string "".

firstTwo("Hello") → "He"

firstTwo("abcdefg") → "ab"

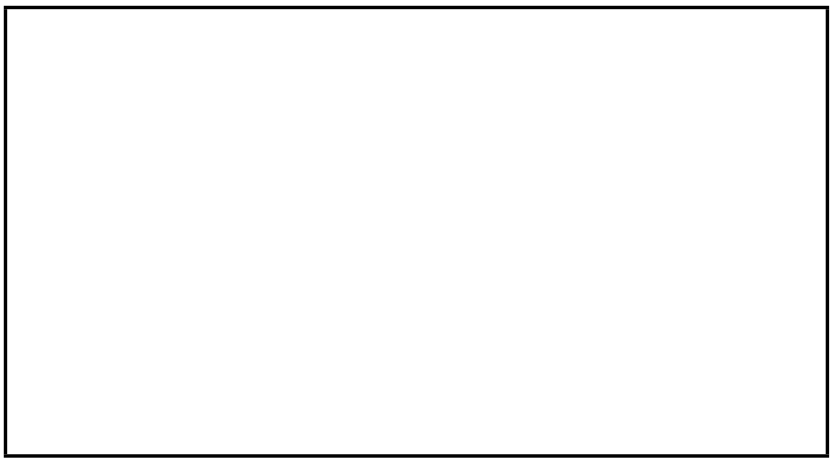
firstTwo("ab") → "ab"

firstTwo("") → ""

The following three questions (4–6) ask you to determine the correct order of the lines to construct a code snippet that achieves a certain purpose. For example, if there are four lines in the code snippet and you think that the first line is d, the second line is b, the third line is a, and the fourth line is c, you need to write the correct order of the lines as d b a c

4. Determine the correct order of the following lines to construct a code snippet that swaps pairs of adjacent elements of the array lst. Note that not all lines are relevant.

```
a.int[]lst= { 1, 2, 3, 4, 5, 6 };
```



```
b.}
c.for(inti= 0;i<lst.length;i+= 2 ){
d.lst[i+1] =temp;
e.lst[i] =lst[i+1];
f.for(inti= 0;i<lst.length;i+= 1 ){
g.inttemp=lst[i];
```

The correct order of lines is: .....

5. Assume that we have an ArrayList of strings called list. Determine the correct order of the following lines to construct a code snippet that receives a sequence of words from the user and fills up the list with distinct words in the sequence. Assume that user enters a blank line to indicate the end of input. Note that not all lines are relevant.

```
a.word=input.nextLine();
b.if( !list.contains(word) )
c.list.add(word);
d.}
e.while( !word.equals("") ) {
f.Scannerinput=newScanner(System.in);
g.List<String>list=newArrayList<String>();
h.While(word!="") {
```

```
i.Stringword=input.nextLine();
```

The correct order of lines is: .....

6. Assume that we have a two-dimensional list of integer called matrix. Determine the correct order of the following lines to construct a code snippet that prints the sum of the elements in each row of the matrix. Note that not all lines are relevant.

```
a.System.out.println(rowTotal);
b.introwTotal= 0;
c.for(intj= 0;j<matrix[0].length;j++ ) {
d.}
e.for(inti= 0;i<matrix.length;i++ ) {
f.}
g.rowTotal=rowTotal+matrix[i][j];
h.rowTotal=rowTotal+matrix[j][i];
```

The correct order of lines is: .....

B.2 PRE- AND POST-TEST IN CLASSROOM STUDY 2

The pre-test is presented below. The post-test was isomorphic to the pre-test: questions 1–5 presented the multiple choices in a different order, questions 6–10 presented the given lines in a different order.

Welcome to the Python Pretest! The test contains 10 questions with a total estimated time of 15-20 minutes. The purpose is to know a bit more about your pre-knowledge about Python programming. Your answers will only be used for research purposes and to improve the course material. This WILL NOT be graded. The test has NO impact on your grades AT ALL. Try your best and thank you!

Please answer the following questions (Q1-Q5) by selecting one of the four options.

Q1. In a bookstore, a $12 book is labeled, "Get a 20% discount". Which one of the following options is the code that calculates the sale price of this book?

☐
```
12 = original_price
0.20 * original_price = discount
original_price - discount = sale_price
```

☐
```
original_price = 12
discount = 20 * original_price
sale_price = original_price - discount
```

☐
```
original_price = 12
discount = 0.20 * original_price
sale_price = original_price - discount
```

☐
```
original_price = 12
discount = original_price / 20
sale_price = discount - original_price
```

Q2. The zoo ticket is free if the person's age is below 15. Otherwise, the price of the ticket depends on the time of the day that we want to visit the zoo. The ticket costs $2 if it's before 1:00pm. The price of the ticket increases to $4 for any other time during the day. Which one of the following options is the code that prints the cost of the zoo ticket for a certain age and hour of the day.

☐
```
if age >= 15:
    if hour < 13:
      print("The ticket costs $2")
    else:
      print("The ticket costs $4")
else:
    print("Free entrance!")
```

□
```
if hour >= 13:
    if age <= 15:
      print("The ticket costs $2")
    else:
      print("The ticket costs $4")
  else:
    print("Free entrance!")
```

□
```
if age >= 15:
    if hour < 13:
      print("The ticket costs $2")
    else:
      if age < 15:
        print("Free entrance!")
      else:
        print("The ticket costs $4")
```

□
```
if age >= 15:
    if hour < 13:
      print("The ticket costs $4")
    else:
      print("The ticket costs $2")
  else:
    print("Free entrance!")
```

Q3. Which one of the following options is the completed version of the following code that aims to find the sum of the integer numbers entered from the keyboard. Assume that user enters 0 to indicate the end of input.

```
sum = 0
num = int(input())
```

```
while ..........
.................
num = int(input())
print(sum)
```

□
```
sum = 0
num = int(input())
while num == 0 :
  sum += num
  num = int(input())
print(sum)
```

□
```
sum = 0
num = int(input())
while num != 0 :
  sum = num
  num = int(input())
print(sum)
```

□
```
sum = 0
num = int(input())
while num != 0 :
  sum += num
  num = int(input())
print(sum)
```

□
```
sum = 0
num = int(input())
while num >= 0 :
  num = num + sum
  num = int(input())
print(sum)
```

Q4. Which one of the following options is the completed version of the following code that aims to increment all values by 1 in the list referred by variable lst.

```
lst = [12, 15, 3, 4, 6, 5]
for ............... :
  ..............
print(lst)
```

☐
```
lst = [12, 15, 3, 4, 6, 5]
for i in range(len(lst)):
  lst[i] += 1
print(lst)
```

☐
```
lst = [12, 15, 3, 4, 6, 5]
for x in lst:
  x += 1
print(lst)
```

☐
```
lst = [12, 15, 3, 4, 6, 5]
for i in range(1,len(lst)):
  lst[i] += 1
print(lst)
```

☐
```
lst = [12, 15, 3, 4, 6, 5]
for i in range(1,len(lst)):
  i += 1
print(lst)
```

Q5. Which one of the following options is the code for the function called firstTwo that receives a string referred by variable s and returns a string made of the first two characters in the string s. So, the string "Hello" yields "He". If the string has less than 2 characters, the function returns whatever the string is. The empty string "" yields the empty string "".

☐
```
def firstTwo(s):
    if len(s) >= 2:
      return s
    else:
      return s[1:3]
```

☐
```
def firstTwo(s):
    if len(s) >= 2:
      return s[0:2]
    else:
      return s
```

☐
```
def firstTwo(s):
    if len(s) >= 2:
      return s[1:2]
    else:
      return s
```

☐
```
def firstTwo(s):
    if len(s) >= 2:
      return s[2:]
    else:
  return s
```

Please answer the following questions (Q6-Q10) by determining the correct order of the lines in the code. For example, if there are four lines in the code and you think that the first line is d, the second line is b, the third line is a, and the fourth line is c, you need to write the correct order of the lines as d b a c. Note that the indentation is not relevant for these questions.

Q6. Determine the correct order of the following lines to construct a code that swaps pairs of adjacent elements of the list referred by variable lst with an even number of

elements. For example, when pairs of adjacent elements of the list [1,2,3,4,5,6] are swapped, the list will be [2,1,4,3,6,5]. Note that you don't necessarily need all lines.

```
a.    lst[i+1] = temp
b.    lst = [ 1, 2, 3, 4, 5, 6 ]
c.    for i in range(0,len(lst),1):
d.    lst[i] = lst[i+1]
f.    temp = lst[i]
g.    for i in range(0,len(lst),2):
h.    temp = lst[i+1]
```

The correct order of lines is: .....

Q7. Assume that we have an empty list referred by variable lst. Determine the correct order of the following lines to construct a code that receives a sequence of words from the user and fills up the list with distinct words in the sequence. Assume that user enters a blank line to indicate the end of input. Note that you don't necessarily need all lines.

```
a.    word = input()
b.    lst.append(word)
c.    if (word in lst) == False:
d.    while word != "":
e.    lst = []
f.    for i in range(len(lst)):
g.    lst = [""] * len(lst)
h.    word = input()
i.    lst[i] =input()
j.    if word != lst:
```

The correct order of lines is: .....

Q8. Consider the class Rectangle defined as follows:

```
class Rectangle:
    def __init__(self, height, width):
        self.height = height
        self.width = width
    def get_height(self):
        return self.height
    def get_width(self):
        return self.width
    def magnify(self, ratio):
        self.height = self.height * ratio
        self.width = self.width * ratio
```

Determine the correct order of the following lines to construct the code that creates a Rectangle object with the height of 10 and width of 20, magnifies the height and width of the rectangle by a factor of three and prints in order the magnified height and width. Note that you don't necessarily need all lines.

```
a.   my_box.magnify(3)
b.   print(my_box.get_height())
c.   my_box = Rectangle(20,10)
d.   print(my_box.get_width())
e.   my_box = Rectangle(10,20)
f.   my_box.magnify(2)
g.   Rectangle.magnify(3)
```

The correct order of lines is: .....

Q9. Assume that we have a two-dimensional list of integer called matrix that is defined as follows:

```
matrix = [[ 1, 2, 3, 4 ],
          [ 5, 6, 7, 8 ],
          [ 9, 10, 11, 12]]
```

You can think of this matrix as a grid of numbers, arranged in 3 rows and 4 columns. The outer list of the matrix contains one inner list for each row, and each inner list contains the integers in one row.

Determine the correct order of the following lines to construct a code that prints the sum of the elements in each row of the matrix, that is: 10 as the sum of the elements in the first row, 26 as the sum of the elements in the second row, and 42 as the sum of the elements in the third row. Note that you don't necessarily need all lines.

```
a. row_total = 0
b. row_total = row_total + matrix[i][j]
c. print(row_total)
d. for j in range(len(matrix[0])):
e. for i in range(len(matrix)):
f. row_total = row_total + matrix[j][i]
```

The correct order of lines is: .....

Q10. Assume that we want to read a text file where each line in the file contains two comma-separated integers. Determine the correct order of the following lines to construct a code that opens up this file and prints out the sum of the two integers in each row.

For example, if the text file contains the following two lines:

12,48 33,11

Then, the code prints

60 44

The code should handle all exceptions that might occur during reading a file. Note that you don't necessarily need all lines.

```
a. print(sum)
b. file = open(filename, "r")
```

```
c. print("Error reading the file. The program execution ends.")
d. sum = int(row[0]) + int(row[1])
e. row = row.split(',')
f. except OSError:
g. file.close()
h. for row in file:
i. try:
j. sum = int(row[0] + row[1])
```

The correct order of lines is: .....

B.3 PRE- AND POST-TEST IN USER STUDY 1

The pre-test is presented below. Questions 1–4 were paper-based. Questions 5–9 were presented on the computer using the PCRS tool (described in Section 2.5). The post-test was isomorphic to the pre-test: questions 1–4 presented the given lines in a different order, questions 5–9 had different surface features.

Q1. Assume that we have two integer variables a and b and we want to set the value of variable result based on the value of a and b. Determine the correct order of the following lines to construct a code snippet that sets the variable result to 1 in either of these two cases: when a is greater than 5 and b is greater than 10 or when both a and b are less than 5. Otherwise, the result will be 2.

```
a.if ((a > 5 || b > 10) && (a < 5 || b < 5)) b. result = 2;
c. result = 1;
d.if ((a > 5 && b > 10) || (a < 5 && b < 5)) e. else
f. int result;
g.if ((a > 10 && b > 5) && (a <= 5 && b <= 5))
```

The correct order of lines is: .....

Q2. Body mass index (BMI) is a measure of body fat based on height and weight that applies to adult men and women. BMI Categories are as follows: Underweight < 8.5

Normal weight = 18.5-24.9

Overweight = 25-29.9

Obesity = BMI of 30 or greater

Determine the correct order of the following lines to construct a code snippet that prints the weight status for the given BMI value.

```
a. System.out.println("Overweight");
b. System.out.println("Normal");
c. else if (bmi < 30)
d. else if (bmi < 25)
e. if (bmi < 18.5)
f. System.out.println("Obese");
g. if (bmi > 18.5)
h. System.out.println("Underweight");
i. else
```

The correct order of lines is: .....

Q3. Determine the correct order of the following lines to construct a code snippet that calculates the sum of every fourth integer starting from 5 up to including 18.

```
a.i = sum + i;
b.for ( int i = 1; i < 18; i += 4 )
c.int sum = 0;
d.sum = sum + i;
e.for ( int i = 5; i < 19; i++ )
```

```
f.sum = i;
g.for ( int i = 5; i < 19; i+=4 )
```

The correct order of lines is: .....

Q4. Determine the correct order of the following lines to construct a code snippet that prints the following output:

```
*****
****
***
**
*
```

```
a.for (int j = 5; j > i; j--) {
b.for (int j = 1; j <= 5-i+1; j++) {
c.}
d.for (int i = 1; i <= 5; i++) {
e.}
f.for (int i = 1; i < 5; i++) {
g.System.out.print ("*");
h.for (int j = 1; j <= 5-i; j++) {
i.System.out.println();
```

The correct order of lines is: .....

Q5. Given integer variables nuts and apples and a boolean variable isWeekend, write a boolean expression to determine if the squirrels have a successful party. The squirrels have a successful party if their party is on the weekend and they have more than either (i) 50 nuts, or (ii) 50 apples to eat. Store the result of this expression in a boolean variable called result.

Assume that the initial value of the variables nuts and apples is already set to an integer and the initial value of variable isWeekend is already set to a boolean value.

E.g. 1: if the value of nuts is 60, the value of apples is 40, and the value of isWeekend is false, the value of result will be false.
E.g. 2: if the value of nuts is 60, the value of apples is 40, and the value of isWeekend is true, the value of result will be true.
E.g. 3: if the value of nuts is 30, the value of apples is 20, and the value of isWeekend is true, the value of result will be false.

Q6. Given three integers a, b, and c, write a code to determine the sum of the three integers. However, if integer a is above 6 or below 4, it does not count toward the sum. Store the sum in an integer variable called sum.
Assume that the initial value of the variables a, b, and c is already set to an integer.

E.g. 1: if the value of a is 4, value of b is 1, and value of c is 2, the value of sum will be 7.
E.g. 2: if the value of a is 7, value of b is 4, and value of c is 1, the value of sum will be 5.
E.g. 3: if the value of a is 1, value of b is 3, and value of c is 7, the value of sum will be 10.

Q7. Given an integer variable age, write a code to determine the fare based on the value of variable age. The fare is $2 for a child (no more than 11 years old), $3 for a senior (at least 65 years old), or $5 for an adult. Store the fare in an integer variable called fare.
Assume that the initial value of the variable age is already set to an integer value.

E.g. 1: if age=7, then fare=2.
E.g. 2: if age=65, then fare=3.

E.g. 3: if age=23, then fare=5.

Q8. Given an integer variable n, write a for loop to print every 5th integer in the range from 5 to n (both inclusive).

Assume that the initial value of the variable n is already set to an integer.

E.g. 1: if the value of n is 10, the code prints: 5↩10↩
E.g. 2: if the value of n is 5, the code prints: 5↩
E.g. 3: if the value of n is 20, the code prints: 5↩10↩15↩20↩
↩ is the newline character.

Q9. Write a nested for loop to print the following triangle with 4 rows:

```
*
**
***
****
```

B.4 PRE- AND POST-TEST IN CLASSROOM STUDY 3

The pre-test is presented below. All questions were presented on the computer. Questions 1–5 were program comprehension question questions and the student had to write the answer for each question in a text field. Questions 6–10 were presented using the PCRS tool (described in Section 2.5). All questions in the post-test were similar to the pre-test but had different surface features.

1. Consider the following code segment:

```
public class Tester {
```

```
  public static void main(String[] args) {

        int result = 9;
        if ( 9 % 2 > 0 )
            result += 2;

  }
}
```

What is the final value of result?

2. Consider the following code segment:

```
public class Tester {
  public static void main(String[] args) {

    int result = 0 ;
    int n = 3;
    while (n > 0) {
        result += n;
        n--;
    }
  }
}
```

What is the final value of result?

3. Consider the following code segment:

```
public class Tester {
  public static void main(String[] args) {
```

```
        int result = 0;
        for (int i = 0 ; i < 5; i++) {
             result = result + i;
        }
  }
}
```

What is the final value of result?

4. Consider the following code segment:

```
public class Tester {
  public static void main(String[] args) {

        int[] numbers = new int[10];
        for (int i = 0; i < numbers.length; i++)
             numbers[i] = i * i;
        int result = numbers[4];

  }
}
```

What is the final value of result?

5. Consider the following code segment:

```
public class Tester {
   public static void main(String[] args) {
```

```
      int[][] matrix = { {0, 2, 5, 0},
                         {1 ,2, 4, 3},
                         {0, 7, 2, 0},
                         {0, 0, 2, 9} };
      int result = 5;
      if (matrix[2][2] > 3) {
            result++;
      }
      System.out.print(result);
  }
}
```

What is the output?
Be careful of the space/newline in your answer.

6. Assume that the program has declared and initialized three integer variables named a, b, and c. Write a boolean expression to determine whether all these three variables have the same value. Store the result of this expression in the variable result.

E.g. 1: if a=5, b=5, and c=5, the value of result will be true.
E.g. 2: if a=4, b=4, and c=7, the value of result will be false.
E.g. 3: if a=3, b=9, and c=3, the value of result will be false.

7. Given an integer variable age, write a code to determine the fare based on the value of variable age. The fare is $2 for a child (no more than 11 years old), $3 for a senior (at least 65 years old), or $5 for an adult. Store the fare in an integer variable called fare.

Assume that the initial value of the variable age is already set to an integer value.

E.g. 1: if age=7, then fare=2.

E.g. 2: if age=65, then fare=3.

E.g. 3: if age=23, then fare=5.

8. Assume that the program has declared and initialized an array of five integers named arr. Write a for loop to increment all values in the array arr by 1.

E.g. 1: if arr=1,2,3,4,5, the code will change it to arr=2,3,4,5,6.

E.g. 2: if arr=10,20,30,40,50, the code will change it to arr=11,21,31,41,51.

9. Write a nested for loop to print the following triangle with 4 rows:

*

**

***

****

10. Write a method called firstTwo that receives a string as its parameter and returns a string made of the first two characters in the given string. If the given string is "Hello", the method returns "He". If the length of the given string is shorter than 2, the method returns whatever the given string is.

Hint: You can use the charAt(int index) or substring(int beginIndex, int endIndex) method for getting the first two characters in the given string.

firstTwo("Hello") → "He"

firstTwo("abcdefg") → "ab"

firstTwo("ab") → "ab"

firstTwo("") → ""

APPENDIX C

SURVEYS

C.1 EXAMPLE EVALUATION SURVEY

The following items were in the survey used in Classroom Study 1 and 3 to evaluate the value of *PCEX* examples. In the survey used in Classroom Study 2, some of these items were slightly modified to point to Python instead of Java.

Part 1. Reasons for low/zero usage

- I preferred to use other resources and material to learn Java
- I was doing well in class without the system and did not need any extra help
- I did not have enough time to use the system
- The system was not introduced properly in class
- I didn't think the system can help me to better master Java
- The user interface was too confusing to use

Part 2. Assessing the impact of examples

- LEARNING
  - Working with the examples-challenges helped me learn Java
  - The explanations in the examples-challenges did NOT help me to better understand the Java programming concepts

  - Exploring similar examples-challenges helped me learn Java
  - The examples-challenges did NOT help me in solving Java exercises in this class

- QUALITY
  - The explanations in the examples-challenges were hard to understand
  - The explanations in the examples-challenges were easy to follow
  - The examples-challenges covered lecture materials reasonably well
  - The codes in examples-challenges were too complicated to understand

- ENGAGEMENT
  - I tried hard to understand the examples-challenges
  - I only skim-read the examples-challenges
  - I thought about how the examples-challenges related to problems I was trying to solve
  - My mind was often wandering to other topics when I was looking at the examples-challenges

C.2 RECOMMENDATION EVALUATION SURVEY

The following items were in the survey used in Classroom Study 3 to evaluate students' experience with the recommendations.

- Perceived recommendation quality
  - I liked the learning materials recommended by the system
  - The recommended learning materials fitted my needs
  - The recommended learning materials were well chosen
  - The recommended learning materials were relevant to my goal
  - The system recommended too many bad learning materials
  - I did not like any of the recommended learning materials

- System satisfaction
  - I would recommend the system to others
  - The system is useless
  - The learning materials that the system recommends are very helpful for me
  - The system recommendations help me have a better practice
  - Using the system is a pleasant experience
  - The system has no real benefit for me

APPENDIX D

VIDEOS

The following videos introduce the practice system that we used in Classroom Study 1 and Classroom Study 2. We named the system as PCLab.

JAVA: https://youtu.be/EGTkrTJ7YaM

Python: https://youtu.be/gv46knva1Lo

BIBLIOGRAPHY

Leona S Aiken, Stephen G West, and Raymond R Reno. *Multiple regression: Testing and interpreting interactions*. Sage, 1991.

Vincent Aleven, Elizabeth A McLaughlin, R Amos Glenn, and Kenneth R Koedinger. Instruction based on adaptive learning technologies. *Handbook of research on learning and instruction. Routledge*, 2016.

Lisa Anthony. *Developing Handwriting-based Intelligent Tutors to Enhance Mathematics Learning*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2008.

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

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

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

Paul Brna. Searching for examples with a programming techniques editor. *CIT. Journal of computing and information technology*, 6(1):13–26, 1998.

Peter Brusilovsky. Explanatory visualization in an educational programming environment: connecting examples with general knowledge. In *International Conference on Human-Computer Interaction*, pages 202–212. Springer, 1994.

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

Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Neidl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 3–53. Springer-Verlag, Berlin Heidelberg New York, 2007.

Peter Brusilovsky and Christoph Peylo. Adaptive and intelligent web-based educational systems. *International Journal of Artificial Intelligence in Education (IJAIED)*, 13: 159–172, 2003.

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

Peter Brusilovsky, Jae-wook Ahn, Tibor Dumitriu, and Michael Yudelson. Adaptive knowledge-based visualization for accessing educational examples. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 142–150. IEEE, 2006.

Peter Brusilovsky, Michael Yudelson, and I-Han Hsiao. 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, 2009.

Kevin Buffardi and Stephen H. Edwards. Introducing codeworkout: An adaptive and social learning environment (abstract only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 724–724, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6.

Michael D Byrne, Richard Catrambone, and John T Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & education*, 33(4):253–278, 1999.

Michael Dwyer Byrne, Richard Catrambone, and John T Stasko. Do algorithm animations aid learning? Technical report, Georgia Institute of Technology, 1996.

Jennifer Campbell, Diane Horton, Michelle Craig, and Paul Gries. Evaluating an inverted cs1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 307–312, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6.

Jennifer Campbell, Diane Horton, and Michelle Craig. Factors for success in online cs1. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 320–325, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4231-5.

ShanaK Carpenter, NicholasJ Cepeda, Doug Rohrer, SeanH Kang, and Harold Pashler. Using spacing to enhance diverse forms of learning: Review of recent research and implications for instruction. *Educational Psychology Review*, 24(3):369–378, August 2012. ISSN 1040-726X.

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

Michelene TH Chi, Miriam Bassok, Matthew W Lewis, Peter Reimann, and Robert Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*, 13(2):145–182, 1989.

Sandra L Christenson, Amy L Reschly, and Cathy Wylie. *Handbook of research on student engagement*. Springer Science & Business Media, 2012.

Cristina Conati and Kurt Vanlehn. Toward computer-based support of meta-cognitive skills: A computational framework to coach self-explanation. *International Journal of Artificial Intelligence in Education (IJAIED)*, 11:389–415, 2000.

Steve Cooper and Mehran Sahami. Reflections on stanford's moocs. *Commun. ACM*, 56(2):28–30, February 2013. ISSN 0001-0782.

Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1995.

Aleksandar Davidovic, James Warren, and Elena Trichina. Learning benefits of structural example-based adaptive tutoring systems. *IEEE Transactions on Education*, 46 (2):241–251, 2003.

Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. Codewrite: Supporting student-driven practice of java. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0500-6.

Stephen H. Edwards and Manuel A. Perez-Quinones. Web-cat: Automatically grading programming assignments. *SIGCSE Bull.*, 40(3):328–328, June 2008. ISSN 0097-8418.

Barbara Ericson, Steven Moore, Briana Morrison, and Mark Guzdial. Usability and usage of interactive features in an online ebook for cs teachers. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, pages 111–120, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3753-3.

Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research*, pages 20–29. ACM, 2017.

Micaela Esteves and António Mendes. 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*, 2003.

Chris Evans and Nicola J Gibbons. The interactivity effect in multimedia learning. *Computers & Education*, 49(4):1147–1160, 2007.

Katherine Wanjiru Getao. An environment to support the use of program examples while learning to program in lisp. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 1015–1016. North-Holland Publishing Co., 1990.

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

Philip J. Guo. Online python tutor: Embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6.

Steven R Hansen, N Hari Narayanan, and Dan Schrimpsher. Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 2(1):10, 2000.

Andrew Head, Elena L Glassman, Björn Hartmann, and Marti A Hearst. Interactive extraction of examples from existing code. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 85. ACM, 2018.

Roya Hosseini and Peter Brusilovsky. Javaparser: A fine-grain concept indexing tool for java problems. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, pages 60–63. University of Pittsburgh, 2013.

Roya Hosseini and Peter Brusilovsky. A study of concept-based similarity approaches for recommending program examples. *New Review of Hypermedia and Multimedia*, 23(3):161–188, 2017.

Roya Hosseini, I-Han Hsiao, Julio Guerra, and Peter Brusilovsky. What should i do next? adaptive sequencing in the context of open social student modeling. In *Design for Teaching and Learning in a Networked World*, pages 155–168. Springer, 2015.

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

David Hovemeyer, Matthew Hertz, Paul Denny, Jaime Spacco, Andrei Papancea, John Stamper, and Kelly Rivers. Cloudcoder: building a community for creating, assigning, evaluating and sharing programming exercises. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 742–742. ACM, 2013.

I-H Hsiao, Sergey Sosnovsky, and Peter Brusilovsky. Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. *Journal of Computer Assisted Learning*, 26(4):270–283, 2010.

Yun Huang, Yanbo Xu, and Peter Brusilovsky. Doing more with less: Student modeling and performance prediction with reduced content models. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 338–349. Springer, 2014.

Christopher D Hundhausen, Sarah A Douglas, and John T Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13 (3):259–290, 2002.

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

Slava Kalyuga and John Sweller. Rapid dynamic assessment of expertise to improve the efficiency of adaptive e-learning. *Educational Technology Research and Development*, 53(3):83–93, 2005.

Slava Kalyuga, Paul Chandler, and John Sweller. Incorporating learner experience into the design of multimedia instruction. *Journal of educational psychology*, 92(1):126, 2000.

Slava Kalyuga, Paul Chandler, Juhani Tuovinen, and John Sweller. When problem solving is superior to studying worked examples. *Journal of educational psychology*, 93(3):579, 2001.

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

Robin H Kay and Liesel Knaack. 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, 2009.

Kandarp Khandwala and Philip J Guo. 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*, page 57. ACM, 2018.

Bart P Knijnenburg, Svetlin Bostandjiev, John O'Donovan, and Alfred Kobsa. Inspectability and control in social recommenders. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 43–50. ACM, 2012a.

BP Knijnenburg, N Rao, and A Kobsa. Experimental materials used in the study on inspectability and control in social recommender systems. *Institute for Software Research, University of California, Irvine*, 2012b.

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

Henry Lieberman. An example-based environment for beginning programmers. In *Artificial intelligence and education*, pages 135–151. Ablex Publishing, Norwood, NJ, 1987.

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

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

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

Yanjin Long and Vincent Aleven. Supporting studentsâĂŹ self-regulated learning with an open learner model in a linear equation tutor. In *International Conference on Artificial Intelligence in Education*, pages 219–228. Springer, 2013.

Michael Mayo and Antonija Mitrovic. Optimising its behaviour with bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education*, 12: 124–153, 2001.

Bruce M McLaren, Sung-Joo Lim, and Kenneth R Koedinger. When and how often should worked examples be given to students? new results and a summary of the current state of research. In *Proceedings of the 30th annual conference of the cognitive science society*, pages 2176–2181, 2008.

Bruce M McLaren, Tamara van Gog, Craig Ganoe, David Yaron, and Michael Karabinos. Exploring the assistance dilemma: Comparing instructional support in examples

and problems. In *International Conference on Intelligent Tutoring Systems*, pages 354–361. Springer, 2014.

Bruce M McLaren, Deanne M Adams, and Richard E Mayer. Delayed learning effects with erroneous examples: a study of learning decimals with a web-based tutor. *International Journal of Artificial Intelligence in Education*, 25(4):520–542, 2015.

Antonija Mitrovic, Stellan Ohlsson, and Devon K Barrow. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education*, 60 (1):264–272, 2013.

Youzou Miyadera, Kunimi Kurasawa, Shoichi Nakamura, Nobuyoshi Yonezawa, and Setsuo Yokoyama. A real-time monitoring system for programming education using a generator of program animation systems. *JCP*, 2(3):12–20, 2007.

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

Kasia Muldner and Cristina Conati. Evaluating a decision-theoretic approach to tailored example selection. In *IJCAI*, pages 483–488, 2007.

Niko Myller. Automatic prediction question generation during program visualization. In *Proceedings of the Fourth Program Visualization Workshop*, 2006.

Amir Shareghi Najar, Antonija Mitrovic, and Bruce M McLaren. Adaptive support versus alternating worked examples and tutored problems: Which leads to better learning? In *User modeling, adaptation, and personalization*, pages 171–182. Springer, 2014.

Amir Shareghi Najar, Antonija Mitrovic, and Bruce M McLaren. 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, 2016.

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

Thomas L Naps, James R Eagan, and Laura L Norton. Jhavé – an environment to actively engage students in web-based algorithm visualizations. In *ACM SIGCSE Bulletin*, volume 32, pages 109–113. ACM, 2000.

Andy Nguyen, Christopher Piech, Jonathan Huang, and Leonidas Guibas. Codewebs: scalable homework search for massive open online programming courses. In *Proceed-*

ings of the 23rd international conference on World wide web, pages 491–502. ACM, 2014.

Timothy J Nokes-Malach, Kurt VanLehn, Daniel M Belenky, Max Lichtenstein, and Gregory Cox. Coordinating principles and examples through analogy and self-explanation. *European Journal of Psychology of Education*, 28(4):1237–1263, 2013.

Jum C Nunnally and Ira H Bernstein. Psychometric theory. 1978.

Fred GWC Paas and Jeroen JG Van Merriënboer. The efficiency of instructional conditions: An approach to combine mental effort and performance measures. *Human factors*, 35(4):737–743, 1993.

Fred GWC Paas and Jeroen JG Van Merriënboer. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of educational psychology*, 86(1):122, 1994.

Jungkook Park, Yeong Hoon Park, Jinhan Kim, Jeongmin Cha, Suin Kim, and Alice Oh. Elicast: embedding interactive exercises in instructional programming screencasts. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, page 58. ACM, 2018.

Nick Parlante. codingbat.com. http://codingbat.com/about.html, 2017. Accessed on Jan 21, 2018.

Dale Parsons and Patricia Haden. 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*, pages 157–163. Australian Computer Society, Inc., 2006.

Philip I Pavlik Jr, Hao Cen, and Kenneth R Koedinger. Performance factors analysis–a new alternative to knowledge tracing. *Online Submission*, 2009.

Chris Piech, Mehran Sahami, Jonathan Huang, and Leonidas Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 195–204. ACM, 2015.

Peter L Pirolli and John R Anderson. The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue canadienne de psychologie*, 39(2):240, 1985.

Thomas W Price, Rui Zhi, and Tiffany Barnes. Hint generation under uncertainty: The effect of hint quality on help-seeking behavior. In *International Conference on Artificial Intelligence in Education*, pages 311–322. Springer, 2017.

Martina A Rau, Vincent Aleven, Nikol Rummel, and Stacie Rohrbach. Sense making alone doesnâĂŹt do it: Fluency matters too! its support for robust learning with multiple representations. In *International Conference on Intelligent Tutoring Systems*, pages 174–184. Springer, 2012.

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

Johnmarshall Reeve and Ching-Mei Tseng. Agency as a fourth aspect of studentsâĂŹ engagement during learning activities. *Contemporary Educational Psychology*, 36(4): 257–267, 2011.

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

Kelly Rivers. *Automated Data-Driven Hint Generation for Learning Programming*. PhD thesis, Carnegie Mellon University, 2017.

Kelly Rivers and Kenneth R Koedinger. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017.

Jorma Sajaniemi and Marja Kuittinen. Program animation based on the roles of variables. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 7–ff. ACM, 2003.

Ron JCM Salden, Vincent AWMM Aleven, Alexander Renkl, and Rolf Schwonke. Worked examples and tutored problem solving: redundant or synergistic forms of support? *Topics in Cognitive Science*, 1(1):203–213, 2009.

Ron JCM Salden, Kenneth R Koedinger, Alexander Renkl, Vincent Aleven, and Bruce M McLaren. Accounting for beneficial effects of worked examples in tutored problem solving. *Educational Psychology Review*, 22(4):379–392, 2010.

Rolf Schwonke, Jörg Wittwer, Vincent Aleven, RJCM Salden, Carmen Krieg, and Alexander Renkl. Can tutored problem solving benefit from faded worked-out examples. In *Proceedings of EuroCogSci*, volume 7, pages 59–64, 2007.

Rolf Schwonke, Alexander Renkl, Carmen Krieg, Jörg Wittwer, Vincent Aleven, and Ron Salden. The worked-example effect: Not an artefact of lousy control conditions. *Computers in Human Behavior*, 25(2):258–266, 2009.

Andrew Sears and Rosalee Wolfe. Visual analysis: adding breadth to a computer graphics course. In *ACM SIGCSE Bulletin*, volume 27, pages 195–198. ACM, 1995.

Rmi Sharrock, Ella Hamonic, Mathias Hiron, and Sebastien Carlier. Codecast: An innovative technology to facilitate teaching and learning computer programming in a c language online course. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, pages 147–148. ACM, 2017. ISBN 978-1-4503-4450-0.

Teemu Sirkiä. A javascript library for visualizing program execution. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, pages 189–190. ACM, 2013.

Teemu Sirkiä and Lassi Haaranen. Improving online learning activity interoperability with acos server. *Software: Practice and Experience*, pages 1657âĂŞ–1676, 2017. ISSN 1097-024X. spe.2492.

Ellen A Skinner, Thomas A Kindermann, and Carrie J Furrer. 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, 2009.

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

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

John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. Cognitive architecture and instructional design. *Educational psychology review*, 10(3):251–296, 1998.

John Gregory Trafton and Brian J Reiser. The contributions of studying examples and solving problems to skill acquisition. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 1017–âĂŞ1022. ACM, 1993.

Kurt Vanlehn. The behavior of tutoring systems. *International journal of artificial intelligence in education*, 16(3):227–265, 2006.

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

Mark Ward and John Sweller. Structuring effective worked examples. *Cognition and instruction*, 7(1):1–39, 1990.

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

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

Gerhard Weber and Antje Mollenberg. Elm-pe: A knowledge-based programming environment for learning lisp. In *Proceedings of ED-MEDIA 1994*, pages 557–562. ERIC, 1994.

Jon Wetzel, Kurt VanLehn, Dillan Butler, Pradeep Chaudhari, Avaneesh Desai, Jingxian Feng, Sachin Grover, Reid Joiner, Mackenzie Kong-Sivert, Vallabh Patade, et al. The design and development of the dragoon intelligent tutoring system for model construction: lessons learned. *Interactive Learning Environments*, 25(3):361–381, 2017.

Michael Yudelson and Peter Brusilovsky. Navex: Providing navigation support for adaptive browsing of annotated code examples. In *AIED*, volume 5, pages 710–717, 2005.

Brad Vander Zanden, David Anderson, Curtis Taylor, Will Davis, and Michael W. Berry. Codeassessor: An interactive, web-based tool for introductory programming. *J. Comput. Sci. Coll.*, 28(2):73–80, December 2012. ISSN 1937-4771.

Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. Facilitating code-writing in pi classes. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 585–590. ACM, 2013.