The Impact of Connecting Worked Examples and Completion Problems for Introductory Programming Practice

Kamil Akhuseyinoglu[1], Aleksandra Klašnja-Milićević[2], and Peter Brusilovsky[1]

[1] University of Pittsburgh, PA, USA `{kaa108,peterb}@pitt.edu`
[2] University of Novi Sad, Novi Sad, Serbia `akm@dmi.uns.ac.rs`

Abstract. Worked examples have consistently demonstrated their value in education, serving as the model solutions for solving specific problem types. Past studies indicate that combining worked examples with practice problems is more effective than providing either problems or examples in isolation. Despite these findings, the exploration of the effects of grouping worked examples and problems for programming practice is limited, especially in learning environments designed for practice. This paper compares two approaches to content organization in a practice system. The first is to explicitly connect the worked examples and completion problems, allowing students to access them in smaller bundles. The other is to deliver the same set of activities separately, but maintain an implicit connection by grouping them under a topic. We examined the effects of these two approaches on student engagement and performance in a semester-long classroom experiment conducted in a CS1 programming course. The results indicate that explicitly connecting worked examples and completion problems increased engagement with the code completion problems and supported problem-solving performance, leading to higher success rates and persistence.

Keywords: Worked examples · computer science education · CS1 · Python programming · open learner model · classroom study

1 Introduction

Researchers and practitioners in the field of Computer Science Education (CSEd) were always among the pioneers of using computers to support learning and teaching. Over several decades, the CSEd community has developed a broad range of interactive learning tools, frequently referred to as “smart learning content" or SLC [2]. Frequently used types of smart learning content include algorithm and program visualization tools [29], worked code examples [5], and various types of problems with automatic assessment [17]. In the early days, each research team focused on developing just one type of SLC and made them available as software to download and install. However, today most types of SLC are implemented as interactive Web content and can be accessed online by students and instructors all over the world [13]. Moreover, the increased popularity of learning management systems (LMS), learning content repositories [6], and integrated practice systems [21,3,12] made it easy to mix multiple types of online SLC under “one roof” by providing access to various types of interactive examples and problems.

The ability to access different types of SLC from one system, without the need to hunt for it through multiple Web sites, considerably increased student engagement with SLC and led to better learning. However, there is still no agreement on how to structure and mix various types of SLC to make learning more efficient. The currently dominated granularity of content grouping is a large *topic*, such as a lecture or a book chapter. Instructors frequently group together links for various examples, demos, and problems that they recommend exploring after each lecture of the course. Textbook authors and publishers offer bundles of learning content associated with each textbook chapter. This topic-level grouping is also used in modern practice systems [3]. However, coarser- and finer-grain grouping is also used frequently. Many instructors post all SLC for their course in one place in LMS while learning repositories [6] and catalogs [14] list together all content for a specific domain like Java, Python, or SQL. In contrast, modern MOOCs piloted small “micro-topics” combining a small video lecture and associated interactive content [11]. Similarly, modern interactive textbooks [10] interleave text and different types of SLC within small subsections. The lack of clear best practice guidelines motivates a research question: “What is the best granularity level to mix and organize diverse types of learning content?”

As a contribution to answering this research question, we performed a semester-long classroom study that compared the traditional topic-level grouping of examples and completion problems in a practice system with an organization that follows the “micro-topic” approach - a direct-access package of a worked example and similar completion problems. The study produced interesting and insightful results including differences in problem-solving performance, persistence, and engagement. In this paper we present the background of our work, present the study and its results, attempt a deeper analysis to explain these results, and draw conclusions for practice and future research.

2 Related Work

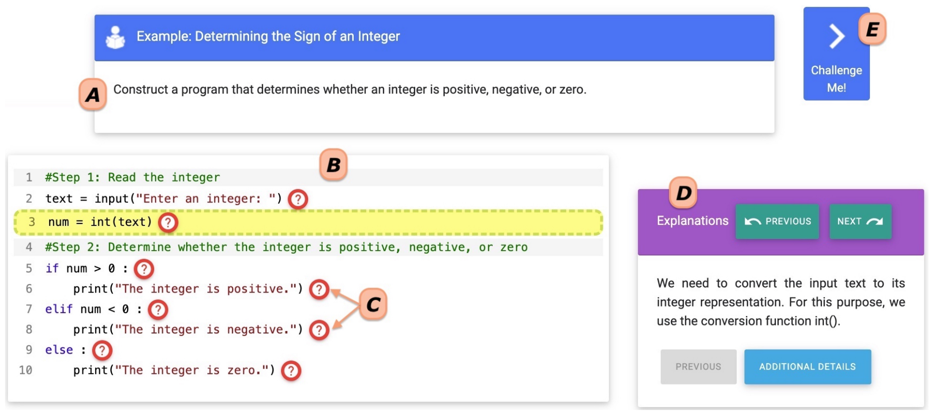
Over the last 20 years, the CS education community has developed and tested a large variety of advanced tools to support teaching and learning programming frequently referred to as interactive or “smart learning content”(SLC) [2] due to their ability to interact with students and provide feedback. Existing types of SLC could roughly be grouped into two large categories: worked examples focused on communicating programming knowledge to the students and problems with automatic assessment engaging students in applying and mastering this knowledge. A large variety of SLC is now available within each category. Among popular types of worked examples are annotated examples [15], code animations [29], and codecasts [27]). Popular types of problems include code tracing problems [4], Parson's problems [24], and coding problems [9]. The unusually large variety of smart content for learning programming could be explained by the need to communicate and practice different types of knowledge, i.e., code understandings [29,4] and code construction [15,17] knowledge as well as by unique features of content types making them valuable in different situations.

While some types of smart content are still used predominantly for assessment purposes (i.e., labs, exams, and homework assignments), collections of smart content are often released for students in a *free practice* mode, i.e., something they can do in their spare time for self-study and self-assessment. Following this trend, two most popular types of practice-oriented systems – interactive programming textbooks [10,26] and practice systems [3,12,21] –typically combine presentation-oriented content (explanations, worked examples) with content focused on self-assessment and problem-solving (questions and various types of problems) offering students opportunities to practice with both types of content. Yet there is still little guidance on how to best organize and structure access to this content and, most importantly, how closely worked examples should be integrated with problems.

Cognitive science research presents convincing evidence that worked examples and problems are equally important for learning each new unit of knowledge; moreover, worked examples should generally precede problems in learning this unit. This is known as *worked example effect*, one of the instructional effects highlighted by the cognitive load theory [19,30]. Several studies in computer science education confirmed this effect and inspired some adaptive systems that recommend worked examples or problems depending on the student's level of knowledge [23,7]. It is not clear, however, how coarse a unit that brings together examples and problems should be: a course, a lecture, a topic, or something even smaller. As mentioned above, the current approaches vary considerably, even within the same group of systems. For example, an online textbook typically places worked examples in the subsections where new concepts illustrated by these examples are introduced. Problems, however, could be placed either right after these worked examples or assembled at the end of each chapter. Does the level of granularity matter, and if so, what is the best way to combine worked examples with self-assessment problems? The study presented in this paper contrasts two levels of granularity for grouping examples and problems for learning programming and attempts to explain the observed differences.

3 Program Construction Examples (PCEX)

Program Construction Examples (PCEX) is an interactive tool to support learning programming construction skills by presenting worked code examples [15]. The tool allows students to explore line-by-line explanations and encourage problem-solving by introducing closely related completion problems. PCEX is designed to deliver a worked example and one to three completion problems, and challenges, as a package with a fixed sequence. We called these completion



Example: Determining the Sign of an Integer

Construct a program that determines whether an integer is positive, negative, or zero.

```
#Step 1: Read the integer
text = input("Enter an integer: ")
num = int(text)
#Step 2: Determine whether the integer is positive, negative, or zero
if num > 0 :
    print("The integer is positive.")
elif num < 0 :
    print("The integer is negative.")
else :
    print("The integer is zero.")
```

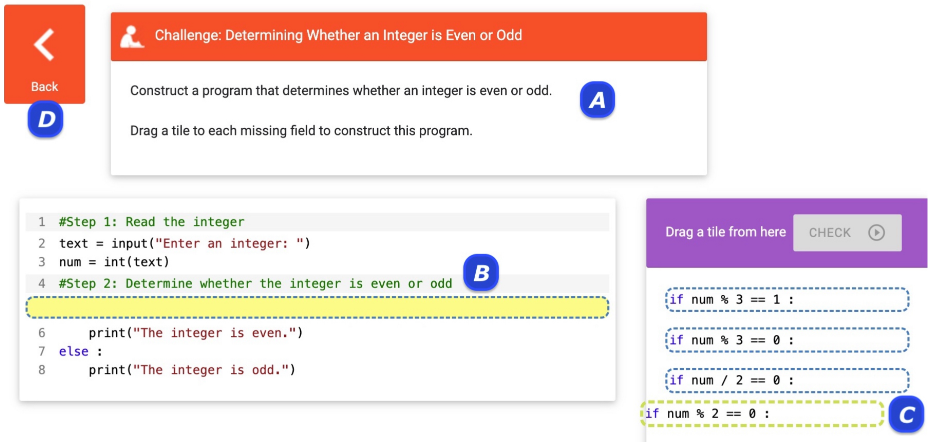
We need to convert the input text to its integer representation. For this purpose, we use the conversion function int().

Fig. 1. A worked example in PCEX. The interface includes a goal description (A), a complete program (B), links to line explanations (C), line explanations (D), and a button to access the challenge (E)

problems “challenges” because they challenge students by engaging them with a problem-solving activity after examining a worked example. The learning activities delivered by the PCEX are authored by an experienced researcher who has strong programming skills and a deep understanding of the introductory programming course curriculum.

As shown in Figure 1, a worked example is designed to address a specific “goal” that is stated on the top (Fig.1A). The goal indicates the expected functionality of the program code presented (Fig.1B). Each worked example has line explanations that summarize “why” that particular line is critical to address the given goal description(Fig.1C). Students could access these explanations by clicking on to question mark icons next to a line and the tool visualizes the explanations in a panel next to the program code (Fig.1D). Students can access to challenges packaged together with the worked example by clicking on the “Challenge Me” button (Fig.1E).

Challenges provide an immediate self-assessment opportunity for students to apply the programming construction knowledge studied in the associated example. A challenge is a completion problem that is closely related to the associated example based on both the goal description and program structure. Figure 2 shows a challenge activity that is accessed after viewing the worked example presented in Figure 1. The goal description of a challenge is presented on top (Fig.2A), and an incomplete code that has one or more missing lines (Fig.2B). To complete the program, students need to drag and drop a code line from the panel on the right (Fig.2C) that includes the correct line and multiple distractors. After completing the program by filling in the missing lines, students can check the correctness of the program and receive immediate feedback. Students can navigate back to the example by clicking on the “Back” button (Fig.2D). If there are multiple challenges associated with an example, students can also navigate between these challenges only after solving the current one correctly or after 3 incorrect attempts.



Challenge: Determining Whether an Integer is Even or Odd

Construct a program that determines whether an integer is even or odd.

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

```
#Step 1: Read the integer
text = input("Enter an integer: ")
num = int(text)
#Step 2: Determine whether the integer is even or odd

    print("The integer is even.")
else :
    print("The integer is odd.")
```

```
if num % 3 == 1 :
if num % 3 == 0 :
if num / 2 == 0 :
if num % 2 == 0 :
```

Fig. 2. A challenge in PCEX that is connected to the example shown in Fig. 1. The challenge contains a goal description (A), an incomplete program (B), code line options (C), and a button to access the related worked example (D).

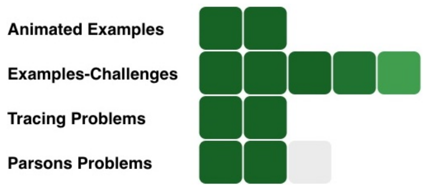
4 Methods

To compare two levels of granularity for organizing worked examples and completion problems (challenges) for learning Python, we conducted a classroom study. In this study, two groups of students were able to practice with work examples and challenges organized in two different ways. One group worked with a traditional topic-level content organization where all worked examples and challenges were listed as separate individual practice activities for the topic (i.e., separate conditions). The other group worked with examples and challenges combined into small packages (i.e., combined conditions), with several of these packages available for each topic. To obtain realistic measures of student engagement with SLC, the study was designed as a semester-long classroom experiment.

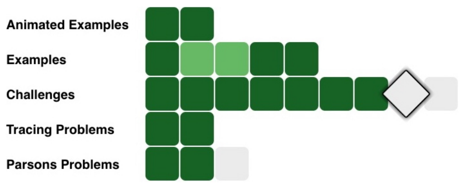
4.1 Study Context

The study was conducted in an introductory Python programming course at a large Australian university. The course does not require any prior programming knowledge and covers programming fundamentals. In the semester of the study, the course had 10 sections delivered online by multiple instructors using the same syllabus, course materials, and grading policy. In total, 338 students were enrolled in the course. The passing grade was 50%, the components of the grade included assignments (30%), class participation (30%), and a project (40%).

Students had access to learning activities through an online practice system. In this study, the practice system had 52 interactive work examples with a total of 530 code line annotations, 71 challenges, 39 animated code examples, 47 code tracing problems, and 34 Parson's problems. The learning activities were organized into 15 topics suggested by the course coordinator to reflect the course design. Within each topic, the activities were further grouped under the different activity types as shown in Figure 3, starting with animated examples, worked examples, challenges (only in separate conditions), tracing, and Parson's problems. Worked examples and challenges were delivered by PCEX. Students could access the activity by selecting a topic to practice with and later clicking on one of the available activities in that topic, shown in a grid design (see Fig.3). Each activity was presented with its completion information, i.e., students could clearly see which activity they completed so far (i.e., green colored cells in Fig.3). The students were also able to observe their topic-by-topic progress in the practice system as well as the average progress of the class through an open learner model (OLM) with personal and social progress-tracking capabilities. Students could get 10% participation credit by solving one tracing and one Parson's problem for each of the 15 topics, i.e., 30 problems in total (37% of the available problems in the system). No credit, however, was given for their work with all kinds of examples, including animated examples, worked code examples, and challenges. The system was introduced to the students by the instructors, following the completion of the first graded assignment.



(a) Combined condition topic view



(b) Separate condition topic view

Fig. 3. Topic views in study conditions showing the learning contents available in the same topic. In (a), students accessed examples and challenges as a package by clicking a cell in the Examples-Challenges row. In (b), students accessed each example and challenge directly by clicking a cell in the Examples or Challenges row.

4.2 Study Conditions

As described in Section 3, PCEX was initially designed to deliver worked examples and challenges as a package. Previously, we evaluated the impact of practicing with the tool on engagement and learning outcomes [15]. We reported that students practiced with the worked examples significantly more when examples were packaged with challenges as compared to a traditional passive style of example presentation (i.e., with no challenges to practice the newly acquired knowledge). In this prior work, students accessed the tool through the same practice system we used for the study presented in this paper which had the same OLM interface. While the packaging approach brought positive results, we were concerned with some negative aspects of packaging. First, we wanted a transparent progress visualization through an OLM, i.e., students' ability to see how much they work with each example or challenge. With packaging, however, the

OLM showed the average progress of a student in the whole package instead of showing their progress in individual activities. Second, due to the encapsulation of challenges in packages, the challenges were effectively hidden from direct view. Students could not even see how many activities were available in the package. With that, we were concerned that they might have overlooked or missed the opportunity to work with the challenges. To overcome these issues, we decided to "open the packages", i.e., provide access to these activities separately making both of them visible on the topic level. We believed that due to the topic-level organization of content in the practice system, eliminating such explicit packaging should not affect students' performance negatively as they would still be able to practice with the examples and challenges together within a topic.

We hypothesized that the separation of worked examples and challenges would lead to higher engagement with challenges (H1a) while keeping students' practice with the worked examples at a comparable level (H1b) without affecting their problem-solving performance (H2) given the OLM design of the practice portal. To test our hypotheses, we designed a classroom study in which students were using the practice systems in one of the following two conditions:

1- Combined Condition: PCEX delivers the learning activities as a package in which a worked example is followed by one to three similar challenges.

2- Separate Condition: PCEX delivers the learning activities as separate activities that allow students to access the worked examples and challenges directly through the topic view provided by the practice system.

In this study, multiple instructors were involved in teaching the course. To ensure consistency, we assigned the same study condition to all sections taught by an instructor. The assignment process involved sorting the 10 sections by enrollment count. The top four sections with the largest enrollments were each assigned to one instructor. Conditions were then assigned based on enrollment, with the largest section receiving the first condition, the second largest receiving the second condition, and so forth. Ultimately, 6 sections were designated for the combined condition, while 4 sections were assigned to the separate condition.

4.3 Data Collection

The practice portal saved all student interaction logs with the learning activities. For problem-type content (challenges, tracing problems, and Parson's problems) the log includes the problem attempt with its result (i.e., correct/incorrect). For example-type content, the log included data on two levels, access to worked example and annotated line views, access to code animation, and animation step views. We analyzed the logs of 190 students who consented to participate in the research study, had interacted with at least one worked example and challenge, and received a final course grade. In addition to the interaction logs, we had access to course grades including individual assignment grades.

5 Results

We reported statistical test results on collected interaction logs after carefully checking the assumptions related to each test. The current study has a hierarchical study design due to how study conditions were assigned to the instructors and sections. Where it was necessary, we conducted linear mixed effects analysis using *lmerTest* packages in *R* [1,20] and included instructor and section identifiers as random effects. To analyze students' problem attempt logs, in which each student had multiple attempts to problems in 15 topics, we also included student, problem, and topic identifiers as random effects to resolve the non-independence of data points in the dataset and control for variance in problem difficulty. We reported LMM results together with the estimated marginal means, not with the observed means. As indicated above, the practice system was introduced after the first assignment. Following this, we included first assignment grades in our regression models for controlling for the prior performance differences. We performed process mining on interaction logs using the bupaR framework [18].

We present the study results following our hypotheses listed in Section 4.2; the effect of the new design on students' practice levels and the changes in problem-solving performance. We also presented a deeper analysis that attempts to uncover the reasons behind the results reported.

5.1 Practice with Worked Examples and Challenges

We began our analysis by examining hypothesis H1, which focuses on how separating challenges from worked examples affects engagement with practice. We first compared *worked example coverage ratio* between conditions, assessing the ratio of explored worked examples to the total available examples. A linear mixed effects model (LMM) revealed that both conditions had comparable worked example coverage ($\beta = -0.17, t = -1.15, p = .35$), after controlling for prior performance (i.e., first assignment grades) ($\beta = 0.19, t = 2.28, p = .02$), even though the example coverage was around 17% higher on average for the combined condition. Second, we checked the differences in annotated line views in worked examples and found that students in the separate condition viewed significantly more lines in ($M = 294$) than combined condition ($M = 143$) ($\beta = 150.9 t = 3.13, p = .03$). However, despite twice as many example lines views, the students in the separate conditions did not spend more time studying worked examples. Vice versa, the students in the combined condition spent marginally more time on worked examples ($M = 10 mins$) than on separate condition ($M = 6.7 mins$) despite the significantly lower number of line views ($\beta = -0.41, t = -1.76, p = .08$). In other words, example exploration was considerably more shallow in the separate condition. Table 1 presents the overall practice statistics for both conditions.

Next, we checked the content engagement differences for challenges. An LMM showed no significant difference in challenge coverage ratio between conditions ($\beta = -0.20, t = -1.70, p = .21$), despite around 15% (10 problems) less coverage by students in the separate condition. However, the total practice time with

Table 1. Summary of practice details in the practice portal per study condition.

	Combined (n=131)			Separate (n=59)		
Measure	Mean	Median	SD	Mean	Median	SD
Example coverage ratio	0.65	0.35	0.77	0.53	0.40	0.48
Line views	134.7	220.5	30.0	311.3	267.1	203.0
Time on examples(mins)	22.0	47.4	10.5	16.6	25.1	7.7
Challenge coverage ratio	0.52	0.35	0.47	0.38	0.37	0.21
Time on challenges(mins)	93.9	84.1	74.1	46.4	52.9	31.7
Session Count	13.4	11.0	8.5	12.1	9.0	9.3
Total coverage ratio	0.63	0.61	0.29	0.51	0.44	0.30
Total practice time(mins)	432.0	392.2	274.0	334.8	262.0	232.0

challenges was significantly longer in the combined condition ($M = 53mins$) than in the separate condition ($M = 21mins$) ($\beta = -0.95, t = -3.12, p = .045$), i.e., the students spent twice as much time on average working on challenges combined with worked examples.

To have a more robust view of differences between the two conditions, we performed additional tests to see whether students had different levels of engagement (i.e., coverage ratios) with other activity types, such as Parson's problems, tracing problems, and animated examples. An LMM revealed no significant differences between conditions indicating that students had comparable levels of activity in other learning items. This suggests that the groups were otherwise comparable in their engagement with practice content and that the observed increase in student engagement with challenges and deeper exploration of examples in the combined condition could be attributed to the package-level organization of this content.

5.2 Problem Solving Performance

We continue our analyses to check the problem-solving performance of students in both conditions, mainly focusing on challenges (H2). We evaluated performance differences by comparing the success rates of students on challenges, i.e., the ratio of correct attempts to total attempts. Thus, we calculated a success rate for every student and the challenge that they attempted. An LMM revealed that students in the combined condition had significantly higher success rates on challenges ($M = 0.32$) than students in the separate condition ($M = 0.25$) ($\beta = -0.07, t = -2.32, p = .022$) while the first assignment grade (a measure of the prior performance) was not a significant factor ($\beta = 0.04, t = 1.21, p = .23$). We also checked students' first-attempt success rates and found that students in the combined condition had solved the challenges more successfully in their first attempts ($M = 0.17$) compared to the separate condition ($M = 0.12$) ($\beta = 0.05, t = -1.99, p = .048$), after controlling for their prior performance ($\beta = 0.04, t = 1.03, p = .30$). A similar effect was not observed for other types of problems, an LMM indicated that students under both conditions had similar success rates ($\beta = -0.01, t = -0.15, p = .88$).

Another important measure of the problem-solving process is the persistence of the students. Persistent students might eventually solve a problem despite initial failed attempts (productive persistence). We labelled each problem as eventually solved or left unsolved, and an LMM showed that the combined delivery of worked examples and challenges led to fewer problems left unsolved than the separate delivery ($\beta = 0.11, t = 2.24, p = .027$) with no significant effect of past performance($\beta = -0.09, t = -1.49, p = .14$). The students in the combined condition were more persistent, leaving 11% fewer problems unsolved.

5.3 How Examples and Challenges Reinforce Each Other?

In previous sections, we showed that students in the combined condition studied worked examples and practiced with challenges much more extensively than the students in the separate condition. Moreover, the packaging of examples and challenges led to higher success rates in challenges on average and on the first attempt, as well as higher persistence that left fewer unsolved challenges. These results suggest that examples and challenges surprisingly reinforced each other when combined into a package. In this section, we performed a deeper analysis to uncover the possible mechanism of this reinforcement.

We performed process mining on student interaction logs focusing on the transitions between worked examples and challenges. We examined the process maps generated for both study conditions and compared the percentage of transitions between specific practice events. Table 2 summarizes the transitional patterns observed in the process maps that we think played a critical role in the observed results.

Table 2. Percentage of transitions observed in the process maps per study condition. Top part of the table shows transitions after a challenge view while the bottom part shows transitions following an incorrect challenge attempt.

Activity Transition	Combined	Separate
Challenge View → Challenge Attempt	68.2	67.0
Challenge View → Example View	**25.1**	**2.3**
Challenge View → Challenge View	3.0	24.4
Challenge View → Other	3.7	6.3
Incorrect Challenge → Correct Challenge	15.0	20.0
Incorrect Challenge → Incorrect Challenge	73.3	77.0
Incorrect Challenge → Example View	**9.8**	**0.6**
Incorrect Challenge → Other	1.9	2.4

We first checked what students decided to perform right after opening a challenge activity. There were four main actions that a student could perform. (1) Make an attempt to solve the challenge (Challenge View → Challenge Attempt), (2) Close the current challenge and open another (or, rarely, the same) challenge (Challenge View → Challenge View), (3) View a worked example (Challenge View → Example View), and (4) close the current challenge and open/attempt another type of activity other than challenge or example (Challenge View →

Other). As highlighted in the first row of Table 2, students in the combined condition decided to go back to the starting example right after seeing a challenge in 25% of the time, while the students in the separate condition performed a transition to an example only in 2.3% of the time.

Similarly, we examined what strategies the student used after an incorrect attempt on a challenge. The bottom part of Table 2 summarizes the transition percentages related to incorrect attempts. Again, students had four options after an incorrect attempt. (1) Make another attempt and solve the challenge (Incorrect Challenge → Correct Challenge), or (2) Make another attempt and fail again (Incorrect Challenge → Incorrect Challenge), (3) viewing a worked example (Incorrect Challenge → Example View), or (4) move to a different type of content to practice (Incorrect Challenge → Other). As summarized in Table 2, in the combined condition students viewed a worked example after failing to solve a challenge in around 10% of all possible transitions. However, students in the separate condition did not (or could not) use this strategy and viewed a worked example only in 0.6% of the cases, 15 times more rarely compared to the combined condition. Instead of finding a helpful example, they tried to solve the challenge again (given a small number of fill-in options to try, it could be a trial-and-error strategy).

5.4 Conceptual Similarity between Examples and Challenges

We further explored the underlying reasons behind the effects of packaged delivery of examples and challenges. Why was providing the same set of activities separately within a topic not as effective as using a combined approach? To explore this issue, we concentrated on the conceptual representation of examples and challenges. We parsed each worked example and challenge to extract a vector of programming concepts used in its code using a Python concept parser tool[3]. The parser tool represents each Python program as a unique set of concepts using an ontology constructed from the Python abstract syntax tree (ABT). Using these conceptual representations, we could measure how similar a specific challenge is to a specific worked example on the concept level by computing the cosine similarity between concept representations. We utilized cosine similarity for its simplicity and we wanted to demonstrate the situation without introducing more complex similarity techniques and measures.

First, we calculated the concept similarity of a challenge to its associated worked example, i.e., the example that was packaged together with that challenge. This similarity represents the situation in the combined condition. Second, we calculated the concept similarity between a challenge and all worked examples available within the same topic. This way we checked how similar a challenge is, on average, to the whole set of examples provided in the same topic, reflecting the case in the separate condition in which students practiced with examples and challenges without an explicit connection. Then, we calculated the average concept similarity per topic following the first, i.e., package-based, and second,

[3] https://acos.cs.hut.fi/python-parser

i.e., separate-based, approach. Finally, we checked statistically significant differences in average similarities between package- and separate-based approaches. A paired Wilcoxon signed rank test revealed that challenges were significantly more similar on concept level to the worked examples in the same package ($M = 0.91$) than to the worked examples within the same topic ($M = 0.79$) ($V = 66, p < .001, r = 0.86$). This provides an evidence that examples and challenges within the same package are more likely to focus on the same programming concepts and thus reinforce each other, than arbitrary pairs of examples and challenges within a topic.

6 Summary and Discussion

We investigated the effects of learning content organization at different levels of granularity by conducting a semester-long classroom study. In one condition, students accessed worked examples and challenges within a coarse-grain topic. In the other condition, the students worked with examples and challenges combined into finer-granularity packages. The results showed that allowing students to access challenges as separate practice items on the topic level did not increase their engagement with the challenges (failed to support H1a). Furthermore, despite extensive study of line explanations under the separate condition, students in the combined condition had a deeper exploration of worked examples (failed to support H1b). The packaging of examples and challenges on a finer-grained level supported student problem-solving performance and led to higher success rates and persistence in challenges (failed to support H2).

Bringing together the results of the study, we could hypothesize that packaging similar examples and challenges together affected student practice with examples and challenges in two different ways. The process mining analysis explained one possible way for the examples and challenges to reinforce each other. In the combined condition, the starting example, i.e., the source of knowledge for solving a specific type of completion problem, was easily accessible by navigating back with a simple click to study it again. We observed that students frequently used this feature and followed a "return to example" strategy when they felt not ready to solve the challenge or failed to solve it. In contrast, although in the separate condition, the same example was also available in the topic, it was hard to locate it among others, so the students almost never tried to do it. Instead, they closed the challenge without an attempt to solve it and opened another challenge, probably, in the hope that it would be easier to solve. The increased persistence rates in the combined condition indicated that going back to the example was, indeed, helpful in gaining the necessary knowledge to solve the challenge.

The second way that likely affected the work with examples and challenges is that closely packaging these types of content decreased student chances to ignore one of these types offering a better opportunity to benefit from their complementary nature. On the one hand, students were forced to encounter an example before trying a group of similar challenges, which gave them the opportunity to be better prepared to solve the problems. In the separate condition, with more freedom of navigation, students could jump to challenges unprepared, without studying an example. On the other hand, separating worked examples from self-assessment problems could cause students to form an "illusion of understanding", i.e., believing they have understood the material upon seeing worked examples and lead to bias in their meta-cognitive judgements. The illusion of understanding is a known issue in learning from worked examples and was observed in several studies [8,25]. In this case, the completion problems placed right after the worked examples could offer an effective and effortless way to engage students in self-assessing their knowledge to check whether the example was really understood [22]. We believe that easy cross-navigation and mutual re-enforcement provided by the packaging of examples and challenges contributed to the differences observed in the study: significantly increased success, persistent rates, and a deeper study of examples.

The remaining question to answer is why the packaged delivery of examples and challenges was helpful for learning while the availability of the same examples and challenges within the topic failed to produce the same effect when they were delivered separately. A possible reason for this difference is that the topic is a relatively large unit of programming knowledge. Each topic usually introduces new programming constructs (i.e., while-loop) and the SLC assembled under the topic typically explains how this construct works and how it could be used to solve typical programming problems. Yet, due to the nature of programming problems, the new constructs could be used in combination with previously learned constructs for solving considerably different programming problems. Based on the previous research it could be argued that typical programming problems introduced through worked examples and challenges represent different programming patterns [28,31], i.e., use programming constructs in different ways and combinations. These patterns represent important subcomponents of programming knowledge to be learned. In this situation, coupling examples and challenges to focus on learning these smaller knowledge components could help in mastering programming knowledge while spreading examples and challenges related to different patterns over the large topic could hinder this process. The conceptual similarity analysis demonstrated that the examples and challenges in small packages concentrates on more closely related programming constructs and reinforce one another, compared to random pairs of examples and challenges within a topic.

To mitigate the adverse effects identified in this study resulting from unpacking examples and challenges, a potential solution is to provide explicit guidance to students similar to the one provided by the packaging, such as reliable content item recommendations. For example, recommending a conceptually similar example after failing to solve a problem or a challenge [16]. Such recommendation techniques should be based on a deeper understanding of programming patterns and the similarity between programming exercises.

Our study is subject to the self-selection bias since the use of examples and challenges was non-mandatory. While we diligently verified our statistical analysis, the study context (i.e., having multiple instructors) led to unbalanced study conditions. We plan to conduct a semester-long, randomized controlled experiment at a different institution to verify our findings. Future work will examine the effects of bundling on additional performance metrics within and outside the practice system. Additionally, future studies might explore another version of PCEX requiring students to write the entire lines of code instead of choosing from multiple options.

References

1. Bates, D., Mächler, M., Bolker, B., Walker, S.: Fitting linear mixed-effects models using lme4. Journal of Statistical Software, Articles **67**(1) (2015)
2. Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirkiä, T., Sosnovsky, S., Urquiza, J., Vihavainen, A., Wollowski, M.: Increasing Adoption of Smart Learning Content for Computer Science Education. In: Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. pp. 31–57 (2014)
3. Brusilovsky, P., Malmi, L., Hosseini, R., Guerra, J., Sirkiä, T., Pollari-Malmi, K.: An integrated practice system for learning programming in python: design and evaluation. Research and Practice in Technology Enhanced Learning **13**(18), 18.1–18.40 (2018)
4. Brusilovsky, P., Sosnovsky, S.: Individualized exercises for self-assessment of programming knowledge: An evaluation of quizpack. ACM Journal on Educational Resources in Computing **5**(3), Article No. 6 (2005)
5. Brusilovsky, P., Yudelson, M.: From webex to navex: Interactive access to annotated program examples. Proceedings of the IEEE **96**(6), 990–999 (2008)
6. Cafolla, R.: Project merlot: Bringing peer review to web-based educational resources. Journal of Technology and Teacher Education **14**(2), 313–323 (2006)
7. Chen, X., Mitrovic, A., Matthews, M.: Learning from worked examples, erroneous examples and problem solving: Towards adaptive selection of learning activities. IEEE Transactions on Learning Technologies **13**(1), 135–149 (2020)
8. Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R.: Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science **13**(2), 145–182 (Apr 1989)
9. Edwards, S.H., Murali, K.P.: Codeworkout: Short programming exercises with built-in data collection. In: 2017 Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE'17. pp. 188–193. ACM Press (2017)
10. Ericson, B., Guzdial, M., Morrison, B., Parker, M., Moldavan, M., Surasani, L.: An ebook for teachers learning cs principles. ACM Inroads **6**(4), 84–86 (2015)
11. Ferguson, R., Sharples, M.: Innovative Pedagogy at Massive Scale: Teaching and Learning in MOOCs, vol. 8719, book section Lecture Notes in Computer Science, pp. 98–111. Springer International Publishing (2014)
12. Gaweda, A.M., Lynch, C.F.: Student practice sessions modeled as icap activity silos. In: 14th International Conference on Educational Data Mining (2021)
13. Guo, P.: Online python tutor: embeddable web-based program visualization for cs education. In: the 44th ACM technical symposium on Computer Science Education (SIGCSE 2016). p. 579–584. ACM Press (2013)
14. Hicks, A., Akhuseyinoglu, K., Shaffer, C., Brusilovsky, P.: Live catalog of smart learning objects for computer science education. In: Sixth SPLICE Workshop Building an Infrastructure for Computer Science Education Research and Practice at Scale at ACM Learning at Scale 2020 (2020)

15. Hosseini, R., Akhuseyinoglu, K., Brusilovsky, P., Malmi, L., Pollari-Malmi, K., Schunn, C., Sirkiä, T.: Improving engagement in program construction examples for learning python programming. International Journal of Artificial Intelligence in Education **30**(2), 299–336 (2020)
16. Hosseini, R., Brusilovsky, P.: A study of concept-based similarity approaches for recommending program examples. New Review of Hypermedia and Multimedia **23**(3), 161–188 (2017)
17. Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O.: Review of recent systems for automatic assessment of programming assignments. In: Koli Calling '10: Proceedings of the 10th Koli Calling International Conference on Computing Education Research. pp. 86–93. ACM (2010)
18. Janssenswillen, G., Depaire, B., Swennen, M., Jans, M., Vanhoof, K.: bupar: Enabling reproducible business process analysis. Knowledge-Based Systems **163**, 927–930 (2019)
19. Kalyuga, S., Chandler, P., Tuovinen, J., Sweller, J.: When problem solving is superior to studying worked examples. Journal of Educational Psychology **93**(3), 579–588 (2001)
20. Kuznetsova, A., Brockhoff, P., Christensen, R.: lmerTest package: Tests in linear mixed effects models. Journal of Statistical Software **82**(13), 1–26 (2017)
21. Laakso, M.J., Kaila, E., Rajala, T.: Ville–collaborative education tool: Designing and utilizing an exercise-based learning environment. Education and Information Technologies **23**(4), 1655–1676 (2018)
22. Mihalca, L., Mengelkamp, C., Schnotz, W., Paas, F.: Completion problems can reduce the illusions of understanding in a computer-based learning environment on genetics. Contemporary Educational Psychology **41**, 157–171 (Apr 2015)
23. Najar, A.S., Mitrovic, A., McLaren, B.M.: 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)
24. Parsons, D., Haden, P.: Parson's programming puzzles: A fun and effective learning tool for first programming courses. In: Proc. of the 8th Australasian Conf. on Computing Education - Volume 52. p. 157–163. ACE '06, Australian Computer Society, Inc., AUS (2006)
25. Renkl, A.: Worked-out examples: Instructional explanations support learning by self-explanations. Learning and Instruction **12**(5), 529–556 (Oct 2002)
26. Shaffer, C.: OpenDSA: An interactive etextbook for computer science courses. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. pp. 5–5. ACM (2016)
27. Sharrock, R., Hamonic, E., Hiron, M., Carlier, S.: Codecast: An innovative technology to facilitate teaching and learning computer programming in a c language online course. In: Proceedings of the 4th ACM Conference on Learning at Scale. pp. 147–148. ACM (2017)
28. Soloway, E.: From problems to programs via plans: The content and structure of knowledge for introductory lisp programming. Journal of Educational Computing Research **1**(2), 157–172 (1985)
29. Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. ACM Transactions on Computing Education **13**(4) (2013)
30. Sweller, J.: Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction **4**(4), 295–312 (Jan 1994)
31. Weber, G.: Episodic learner modeling. Cognitive Science **20**(2), 195–236 (1996)